FALCON: FRAMEWORK FOR ANOMALY DETECTION IN INDUSTRIAL CONTROL SYSTEMS

by

Subin Sapkota

A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Computer Science

Boise State University

December 2019

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the thesis submitted by

Subin Sapkota

Thesis Title: FALCON: Framework for Anomaly Detection in Industrial Control Systems

Date of Final Oral Examination: 22nd October 2019

The following individuals read and discussed the thesis submitted by student Subin Sapkota, and they evaluated the presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Dr. Hoda Mehrpouyan, Ph.D.	Chair, Supervisory Committee
Dr. Casey Kennington, Ph.D.	Member, Supervisory Committee
Dr. Cathie Olschanowsky, Ph.D.	Member, Supervisory Committee
Stephen Reese, PE	Member, Supervisory Committee

The final reading approval of the thesis was granted by Dr. Hoda Mehrpouyan, Ph.D., Chair of the Supervisory Committee. The thesis was approved by the Graduate College.

Dedicated to my parents and my wife.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to the Boise State University Graduate College and Computer Science Department for their academic support on my graduate school journey. I would also like to express my gratitude to National Science Foundation (NSF) award number 1846493 that supported the work in this thesis.

I am grateful to Dr. Hoda Mehrpouyan for taking me in as her graduate assistant and supporting me throughout the two years in Boise State University.

I would also like to thank my committee members, Dr. Cathie Olschanowsky, Dr. Casey Kennington, and Stephen Reese (Idaho National Laboratory), for their comments and guidance that have helped shape the scope and experiments within this thesis.

I am forever grateful to my parents, Bimala Mishra and Sudarshan Sapkota, for their everlasting love and support during my academic journey. I am also grateful to my in-laws, Stephanie Hill-Lang and Joseph Lang, for their constant support.

Finally, I would like to thank my wife, Kaitlyn Lang, for being the support through all the ups and downs of graduate school and the thesis writing process.

ABSTRACT

Industrial Control Systems (ICS) are used to control physical processes in the nation's critical infrastructures. They are composed of subsystems that control physical processes by analyzing the information received from the sensors. Based on the state of the process, the controller issues control commands to the actuators. These systems are utilized in a wide variety of operations such as water treatment plants, power, and manufacturing, etc. While the safety and security of these systems are of high concern, recent reports have shown an increase in targeted attacks that are aimed at manipulating the physical processes to cause catastrophic consequences. This emphasizes the need for algorithms and tools that provide resilient and smart attack detection, as well as risk analysis mechanisms to protect the ICS.

To address this need for resiliency, this thesis designs and develops an anomaly detection and risk analysis framework for ICS. The proposed anomaly detection methodology utilizes dilated Convolution and Long-Short Term Memory (LSTM) layers to learn temporal as well as long term dependencies from sensors/actuators data in ICS. This data is passed through a unique feature engineering pipeline where wavelet transformation is utilized on the sensor signals to extract additional features. Additionally, this thesis explores four different variations of supervised deep learning models, as well as an unsupervised one class Support Vector Machine (SVM) model for this problem. Furthermore, an empirical analysis of a single monolithic model for all sensors/actuators in ICS vs distributed models for each segmented process is carried out.

The proposed methodology is validated utilizing sensors/actuators normal and attack data from a miniature water treatment plant known as Secure Water Treatment (SWaT) testbed. The results of our experiments show improvement over existing state-of-the-art anomaly detection algorithms with higher performance than the baselines set previously. In addition, this thesis provides evidence on monolithic models trained on entire processes in ICS performing better than the distributed models due to their ability to learn global relationships within the data. Along with an anomaly detection methodology, this thesis also presents a Colored Petri Net (PN) model for simulating the physical processes based on control code, and modeling risks within the system.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

Industrial Control Systems (ICS) are used to control physical processes in industries and critical infrastructures. ICS are composed of sub-modules that control physical processes in industries by analyzing the information received from the sensors. Based on the information about the state of the process received from sensors, the controllers in ICS issues a control command to the actuators. These control systems are used in a wide variety of operations such as nuclear power plants, manufacturing, oil and gas pipelines, water distribution systems, etc. Therefore, they are integral to the safety and security of a nation's critical infrastructure. As a result, initiatives such as the United States of America Presidential Executive Order No. 13,636 (2013), have been passed to protect the critical infrastructures of a nation including the ICS that control them [40]. Due to their critical nature, attacks on ICS that propagate to the physical processes can cause large monetary losses, environmental damages, and significantly impact the infrastructure of cities. These significant impacts have been demonstrated by well-known attacks such as the Ukraine power blackout [9], Stuxnet worm [14], and the Maroochy Shire Sewage Spill [1]. Furthermore, the Cybersecurity and Infrastructure Security Agency (CISA) publicly reported that the number of attacks against ICS has steadily grown in 2016 alone with 290 reported attacks [36].

This increase in the targeted attacks on ICS emphasizes the fact that we need algorithms and tools that provide resilient and smart attack detection mechanisms coupled with risk analysis to protect ICS.

As a result, many researchers have started focusing on techniques that help with anomaly detection within ICS. The existing state-of-the-art research can be divided into two categories of: (I) Formal methods using behavior specification and verification [33, 29, 3, 15], and (II) Data driven approaches: (a) Machine Learning (ML) [28, 11], (b) Deep Learning (DL) [23, 27, 42, 38], as well as (c) statistical modelling [4].

Formal logic verification approaches test for deviations of signals from systems' requirements based on formal models of systems built using various symbolic grammars and invariant logic. While these models are robust due to their use of design specifications of system, they are susceptible to state explosion problems due to the high number of specifications and components within ICS [33]. At the same time, complete verification engines within the controllers code is not feasible due to the low capacity of embedded controllers. Also, constructing a model of specifications of exact behavior of ICS is an expensive process as it requires precise understanding of ICS physics, and this understanding is hard to obtain for even current domain experts [29].

Another approach to ICS anomaly detection utilizes Machine Learning to detect the deviations from normal behavior in the specified feature space. Their shortcomings are their failure to detect novel attacks not observed by the model. In addition to Machine Learning, Deep Learning is used to learn the normal behavior of the system from the anomalous behavior utilizing the historical data. Current research in this area are only able to capture a small subset of attacks, and have expensive training

time as well as cost due to the large number of features and data that need to be analyzed and computed. Signature based statistical modelling is another approach that is based on creating noise signatures for sensors and detecting any deviations from the created signatures as anomalies. This approach is highly susceptible to noise in the operating environment, and limits it to small number of devices in the process [4]. Therefore, there is need for an anomaly detection methodology in ICS that can overcome the challenges faced by these previous works.

In addition to anomaly detection, risk analysis of physical processes helps promote resilient system design and smart monitoring. Resiliency can be achieved through the study of potential attacks as well as attacker behaviors within a system, and adding security measures for risks learned from this study. In addition, smart monitoring can be carried out by identifying potential attack propagation and highly vulnerable points within a system. In previous works, these goals have been achieved using three different methodologies: Game Theory [34, 5], Attack Trees [8], and Petri Nets [24]. While each approach presents an unique finding and methodology to analyse risks in a system, they fail to provide risk analysis for attacks and their impacts within the physical processes controlled by ICS. Therefore, there is need for modelling, simulation, and risk analysis of attacks within the physical processes controlled by ICS to identify potential vulnerabilities as well as attack propagation paths.

Therefore, this thesis aims to to overcome the challenges within the research of security in ICS. The following section presents the thesis statements, and the research questions to be answered.

1.2 Thesis Statement

The goal of this research is to increase the resiliency of ICS to cyber-physical attacks by:

- Accurately detecting anomalous behavior in sensors and actuators in real-time,
- Identifying the most efficient architecture for training and deployment of anomaly detection model, and
- Defining a physical process simulation and risk analysis methodology.

1.3 Research Questions

This research aims to achieve the goals by answering the following research questions:

- *RQ 1:* How to quickly and accurately identify anomalous behaviors resulting from cyber-physical attacks in sensors and actuators within Industrial Control Systems (ICS)?
- *RQ 2:* How to empirically determine the relative effectiveness of different architectural designs using metrics such as percentage of attacks successfully detected, and training time?
- *RQ 3:* How to design and develop risk analysis methodology to investigate the effects of attacks within the ICS controlled processes?

CHAPTER 2

ANOMALY DETECTION USING NEURAL NETWORK

2.1 Related Work

A large portion of studies on the protection of ICS have focused on protection of the Information Technology (IT) systems within the industries [37, 10, 7, 17]. While the protection of IT infrastructure is important, the resiliency of control systems to cyber-physical attacks requires anomaly detection focused on observing physical processes of ICS [2]. In this section, we classify studies on anomaly detection for ICS into two categories: Formal Models, and Data Driven Models. The rest of this section provides a detailed review of different approaches.

Formal Models for security

Current work in this field focuses on verification of security properties based on mathematical proofs that system specifications are met. These model properties are built based on existing system specifications, or specification mining based on historical logs of the system. Once formal models are built and violations of specifications are captured, they are used as invariants for detection of anomalies.

Application of formal models, through symbolic grammar definition, is used for anomaly detection in ICS by Kang et al., [29]. Alloy modeling language [31] is used to prepare a first-order linear temporal model for the system based on engineering

schematics. This model captures all process state transitions, as well as security properties that are to be satisfied for the system. Along with the system model, an attack specification is also modelled using Alloy language. The captured models are then run through an Alloy analyzer to find counter-examples that invalidate the safety properties. Counter-examples are system traces, in terms of state changes, that result in an unsafe state or exceed operational boundaries defined in the system specification. These counter-examples are then validated by running the system trace described by them. Domain experts then mark the counter examples as invalid/valid based on their effect in the test-bed. Valid counter-examples are then used for re-building the system specification and as examples to detect anomalous behavior.

This system is limited by the knowledge overhead it introduces on engineers, and the number of parameter tuning it requires. It also assumes that security properties defined in system specifications, attack specifications defined by engineers, and the system behavior defined in engineering schematics are complete and can be used as ground truth. In addition, the values of each continuous component[1] are concretized to analog behavior by engineers, introducing false vulnerabilities and can miss other vulnerabilities within the system.

A different approach to using system specifications for rule generation is proposed by Adepu et al., [3]. Here, invariants are manually derived from the mathematical relationships that exist between components of a system that are controlled by unique controllers. State Agnostic and State-Dependent invariants are built and coded into individual controllers for monitoring. These invariants represent states of components as nodes, and transitions between them as edges. Any behavior that does not follow

[1]For the remainder of the chapter, components/features are interchangeably used to refer to sensors and actuators.

the defined invariants are classified as anomalous.

Limitations of this approach lie in the requirement of discretization of values for definition of states, losing information on components of the system. They also fail to provide results of the tests on attacks generated. Another limitation of this approach is the need to code invariants directly into controllers. This requires the detection mechanisms to be limited by the memory within controllers. Also, invariants checking cannot be competing with resources with the controller as they need to function in real-time. The proposed method does not address this problem.

Another use of formal logic, through rule based representation of system, is proposed by Fauri et al., [15] where domain experts parse network traffic between controller and components of the system, and label important variables within them. These labeled feature values are then binned based on their occurrences during normal operation of the system. A single state of the system is classified as anomalous if the probability of a state occurring in normal system bins is below a user-defined threshold.

This method is limited by the complexity of labeling and contextualization of system events to variables. This process can leave a large margin for error depending on how well the network traffic is visualized to engineers during process of labelling. Domain experts are required to perform intervention after detection to identify detected anomaly as True Positive or False Positive, and make online changes to the model.

To combine rule based models and symbolic grammar models of formal models, Tabor et al. [33] propose using Probabilistic Deterministic Real Timed Automaton (PDRTA) to learn and build a transition model for a system. Here, sensor signals are discretized and states of system based on sensor signals are specified in terms of

timed strings. Once the timed strings are obtained, a PDRTA is learned using RTI+ algorithm. This automaton represents the states of system as nodes and transitions as edges on an acyclic graph. Finally, any system behavior that does not appear in the PDRTA is flagged as anomalous.

This method is limited by sensors signals segmentation and alignment. This process can introduce large complexity due to the noise in sensor signals, and discretization of continuous values of signals. At the same time, PDRTA only learns a single component model and fails to learn relationships between components within a system.

Data Driven Models

In this section, we describe the Machine Learning and Deep Learning techniques explored to tackle the challenge of anomaly detection in ICS. Whereas formal logic models use system specification to find behaviors that violate security models, data driven approaches use input of data obtained from ICS to learn an anomaly detection model that classifies deviations from normal feature space as anomalous.

Machine Learning

Effectiveness of variety of Machine Learning (ML) algorithms for anomaly detection in ICS is explored by Junejo et al., [28]. They utilize labeled attack and normal data from SWaT test bed [22] to train classification models using various ML algorithms. They train algorithms such as Support Vector Machine (SVM), Neural Networks (NN), Instance-Based Learning (IBK), Decision Trees, Namely Random Forest (RF), J48, Best-First Tree (BFTree), Naive Bayes (NB), Bayesian Network (BayesNet), and Multinomial Logistic Regression (LR) on a single process, $P1$, of the test-bed. This

method is limited on the information about the anomalous behavior it can provide to the engineers once an anomaly is detected. At the same time, detection of unknown attacks whose behaviors are not reflected in training data is not possible with this methodology. Finally, specifications on obtaining labeled attack data for systems are not explored.

Another approach to ML is proposed by Chen et al. [11]. Here, simulations built to mimic ICS are leveraged to produce attack data for code mutation attacks. Initially, codes are exposed to permutations of mutations, and mutated codes and their outputs are labeled as anomalous in order to build the data. The obtained data is then fed into an SVM algorithm to build a two class attack classification model. This work is limited to code mutation attacks, and the extensible nature of this approach to other attacks is not explored. Furthermore, the permutations of mutations built on simulations cannot capture all deviations from the normal behavior of the system.

Deep Learning

Apart from ML, unsupervised learning using Neural Networks are explored to find anomalous behaviors that deviate from normal feature space without the need of pre-existing attack signatures.

Initially, SANDIA Laboratories proposed the Weaselboard framework [38] to detect novel vulnerabilities utilizing Bayesian classifiers. This method was limited by their assumption of independence between variables of ICS, which is invalidated by the connectivity and dependencies of sensors, actuators, and inter-connected processes. More importantly, details of the implementation of the system are lacking due to their intellectual property rights.

After the introduction of bayesian network, Goh et al., [23] introduced the use of Long Short Term Memory-Recurrent NN (LSTM-RNN) on a single process of the SWaT data [22]. While powerful, their experiment is limited to a single process of SWaT data. Even for this small feature space, it took the model a single day to train while detecting 9/10 of attacks. At the same time, they also point to 4 False Positives (FP) detected. These FP's are not based on comparison with labels within the available attack dataset, but manual study of attacks and distance of alarm raised to any attack period. Also, scalability of the model when training for all features is left unexplored.

Another approach to Deep Learning is explored by Inoune et al., [27] where a probabilistic outlier detection using Deep Neural Network (DNN) is used to produce an outlier factor value for a time window of data. They apply feature engineering to the data, where instead of training on the time-sampled data, probability distribution of the time windows are calculated, and the probability of occurrence is used to train the network on all processes of the SWaT testbed.

While powerful, training of this model had a time cost of 2 weeks, and testing for entire data had a time cost of 8 hours. An in-depth view of recall on all attacks shows that the model has 0 recall value for 23/36 attacks. This is a significant gap in attack detection for attacks carried out in the testbed.

While different architectures for training NN for anomaly detection were approached, Shalyga et al., [42] explored using Genetic Algorithm (GA) to find a neural network architecture that performs the best forecasting on the SWaT data. They also explored the usage of exponentially weighted moving average smoothing (EWMA), mean p-powered error measure, weighted p-powered error, and disjoint prediction windows to improve the quality of anomaly detection.

Their approach to building a forecasting model, while novel, introduced post-processing of errors that increased anomaly classification time. This increase in time is especially observable on classifications that took over 1000 seconds. These long delays in detection do not apply well to ICS domain where anomalies need to be detected quickly and real-time mitigation measures need to be taken.

Statistical Modelling

Another area of data driven approach to anomaly detection is using statistical modelling to create identification of system components. Application of this approach is proposed by Ahmed et al., [4] where identity of sensors is created based on a deterministic model of their noise. To build the deterministic model, all sensors are assumed to have a unique noise due to their build themselves. This noise captured is assumed to be unique for different sensors. This assumption is used to model sensor noise patterns and detect sensor swap or replacement attacks by comparing stored patterns for each sensor.

This methodology has a fundamental flaw in that sensors are affected by environmental noise as well as internal noise. ICS is not isolated from the environment, and hence sensors have varying degrees of noise. Authors have not addressed the problem of robustness of the model to outside noise. It is also limited to detecting two specific attacks for sensors, and actuation components are not explored.

Limitations of Anomaly Detection in ICS As explored in this section, formal models are limited by their introduction of knowledge overhead, state explosion problem when modelling a large system, and dependence on completeness of system security specifications. At the same time, data driven approaches have been limited

in their recall of attacks, and timely detection of attacks. They are also exposed to a large number of false positives. On top of this, implementation details of architectures proposed are limited to a single testbed, and in some cases limited to a single small process. In some cases, introduction of manual concretization of features for statistical and formal modeling have introduced loss of information in models. In this research, we aim to focus on answering our *RQ 1* to build an anomaly detection system that can detect anomalies accurately and in real-time. This will aim to fill the gap of data driven approaches that are limited in recall of attacks, and timely detection of attacks.

In this thesis, we utilize a data driven methodology for building an anomaly detection model. Sensors and actuators data from ICS is passed through a pre-processing pipeline where sensors values are decomposed using wavelet transformation and added as features for training. A supervised neural network anomaly detection model is trained and improves upon the performances of previous works explored in this domain. To our knowledge, this is the first implementation of a LSTM and CNN hybrid architecture shown in this domain, and performs better than other supervised as well as semi-supervised methodologies by detecting 27 attacks, and completing training in $20 - 30$ minutes

2.2 Methodology

This section will explain the feature engineering pipeline, architecture components, and the anomaly detection framework in detail.

2.2.1 Data

Data for the anomaly detection framework has multiple features, representing all sensors and actuators in the system, collected throughout several days with a specific time-step. Features are snapshots of values of all sensors and actuators within an ICS taken for a time-step. Since ICS processes status change gradually and in real-time, samples are recorded with the time-step of one second. These samples are recorded over multiple days to collect the large amount of data required for training anomaly detection models. Data collected in this process do not need adherence to a specific range of values as further processing will be carried out before their usage.

Let us define this data format. Let, a set $S = \{s_0, s_1, ..., s_n\}$ represent the n number of sensors and actuators within a system being controlled by ICS. Then, $V_i = [s_{0i}, s_{1i}, ..., s_{ni}]$ represents the values of all sensors and actuators sampled at a time i. For readability when porting to the code base, the start time of data sampling is represented as $i = 0$. Finally, $D = [V_0, V_1, ..., V_{\frac{(t*86400)}{ts}}]$ represents the matrix of values of all sensor and actuators sampled over a uniform time-step of ts seconds for a time-period of t days. With this representation, we will have collected $m \times n$ matrix of data where $m = \frac{(t*86400)}{ts}$.

This collected matrix of data will then be pre-processed before being used for the training of the forecasting model. Our pre-processing consists of:

- Cleanup of missing or interpolated values,
- Scaling of continuous data to a range of $[0, 1]$,
- Divide data into a sliding-window format with window size (ws) as a parameter to be heuristically optimized for,

- Feature Engineering to append information into existing data,
- Split data into train and test sets for the training of the models.

2.2.2 Feature Engineering

Pre-processing of data is required in order to overcome the challenges that high dimensional data impose on attack detection components in ICS. Since our proposed anomaly detection framework is faced with large amount of high dimensional sensors and actuators' data along with continuously evolving attack characteristics, we first use the scaling range $[0, 1]$ to uniformly scale all values. Further, a Wavelet Transformation [12] is utilized to transform sensor signals and obtain temporal information on multiple scales. Wavelet Transformation is an invertible transformation that can be used to decompose a signal into a high and low spectrum using high and low pass filters respectively. The low pass spectrum is a down-sampled decomposition of the original signal. The high pass spectrum is the localization of higher frequencies of the original signal. These decomposition's are generated by equation (2.1)

$$\begin{aligned} low_k &= \sum_m g(2k-m)x_m \\ high_k &= \sum_m h(2k-m)x_m \end{aligned} \tag{2.1}$$

where k is the coefficients index, $h(.)$ and $g(.)$ are low and high pass filters built from a mother wavelet function. Once the decomposition coefficients are calculated, we append the low and high spectrum decomposition to the existing feature set of data. In this context, wavelet transformations are preferred over Fourier transformations because of their ability to keep temporal locality information of the signal frequencies. For our purpose, we use a type of Wavelet Transformation that is sensitive

to alignment of signal in time called Daubechies Discrete Wavelet Transformation (DWT) [12]. In order to keep the original signal coefficients within the transformed low and high spectrum output, we apply a single level DWT transformation.

After the wavelet transformation, we label each time-step of sensor/actuator values as attack/normal behavior. If a time step contains anomalous, faulty, or attack data it is manually labelled as an attack instance. Further, the result is converted into a sliding window format where each window size is a hyper-parameter. For each sliding window containing an attack label, the entire window is labelled as an attack behavior. The next step in the proposed framework is to train a classification model for the purpose of anomaly detection.

2.2.3 Architecture Components

As mentioned in Section 2.1, previous works utilizing deep and machine learning for anomaly detection within ICS have explored techniques such as Long Short Term Memory (LSTM), Support Vector Machine (SVM), Naive Bayes Classifiers, etc. Each of the methodologies mentioned above were explored in this research work, but none of them performed as well as our final model. Therefore, in this chapter, we utilize a supervised Deep Neural Network methodology to design a two-class attack detection model for ICS time-series sensor/actuator data. To construct an effective anomaly detection model, the proposed framework relies on Convolutions and Long Short Term Memory (LSTM) based layers as described below:

Long short term memory networks (LSTM)

LSTM, which is the extended form of RNN, is used to detect long term temporal patterns in sensor/actuator's time series data. Given an input signal $x = (x_1, x_2, ..., x_T)$

where T is the length of the signal, LSTM hidden state at timestep t is computed by:

$$Input\ Gate\ : i_t = \sigma(W_i\dot{[}h_{t-1}, x_t] + b_i) \tag{2.2}$$

$$Forget\ Gate\ : f_t = \sigma(W_f\dot{[}h_{t-1}, x_t] + b^f) \tag{2.3}$$

$$Output\ Gate\ : o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.4}$$

$$Cell\ Input\ : g_t = tanh(W_c\dot{[}h_{t-1}, x_t] + b_c) \tag{2.5}$$

$$Cell\ State\ : c_t = f_t \bigodot c_{t-1} + i_t \bigodot g_t \tag{2.6}$$

$$Cell\ Output\ : h_t = o_t \bigodot tanh(c_t) \tag{2.7}$$

where $\sigma(.)\ and\ tanh(.)$ are the element-wise sigmoid and hyperbolic tangent functions and $\bigodot$ is the element-wise multiplication operator. h_{t-1} and c_{t-1} are the hidden state and memory cell of previous time-step. As depicted in equations (1)-(4), i_t, f_t, o_t, and g_t are input gates, forget gates, output gates, and cell states. During each iteration of training, the LSTM layer needs to decide information to be thrown from the stored cell states by calculating the forget gate (f_t) for each point in the cell state c_t. Next, the input gate decides on values to be updated, while g_t creates a candidate that could be input into the cell state. Finally, the old cell state is updated by calculating c_t by using the outputs of the forget gate (f_t), input gate (i_t), and the cell state candidate (g_t). Once the cell states have been updated, the output from the cells need to be calculated. An output gate (o_t) decides on parts of the cell to be used for the output, and this is multiplied with a *tanh* value of the cell state to calculate the final output from the recurrent unit.

As it is depicted in the above formulas, the LSTM model requires storage of additional cell states in the memory and needs to compute these cell states. While LSTM's ability to extract long-term temporal dependencies without suffering from

the vanishing gradient problem [20] helps with an effective anomaly detection, it is considered a relatively computationally expensive model [6]. The work-around is to design an efficient neural network architecture to balance the utility of LSTM while minimizing the computational overhead caused by it. This can be achieved through the reduction of the size and depth of LSTM layers in the proposed neural network architecture.

Convolutional Neural Network (CNN)

Convolutions are known to be used to extract features for normal/attack classification of data [46, 18]. Given an input signal, a convolution layer with m filters and a filter size of n extracts a m-dimension feature vector from every n values in the signal. These filters can extract temporal locality from inputs, and help learn temporal dependencies in a multi-variate time-series input datalecun1995Convolutional. Therefore, 1D Convolutions can efficiently learn temporal and spatial dependencies within the time-series sensors and actuators input data of ICS.

In this work, we utilize a specific type of Convolution layer known as dilated causal Convolution layer. This type of Convolution layer adds two unique properties of dilation and causal padding to the Convolution layer. The causal property refers to a Convolution layer where $output[t]$ does not depend on the inputs from the future ($input[t+1:]$). This property helps preserve the temporal order of the sequential time-series data within our model [44]. In addition, with the added dilation property, we are able to construct a more compact CNN model and still preserve the global relationships between the features. This is feasible because dilations allow each hidden feature in CNN to observe increasingly larger input space (receptive field) as the depth

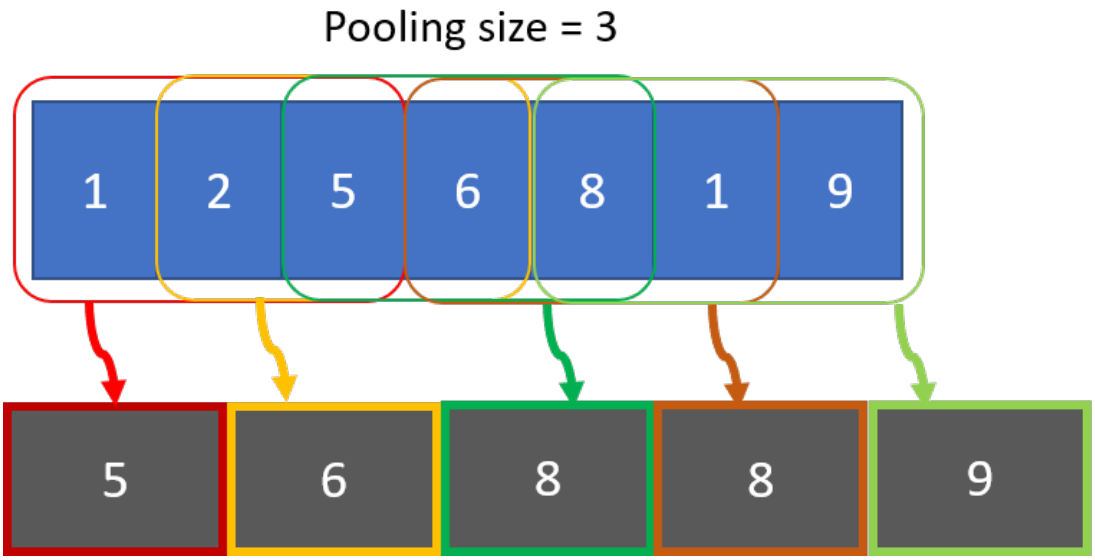


Figure 2.1: 1D MaxPooling with pooling window size of 3.

of the CNN increases. Hence, it allows the increase in the receptive fields of the layers without the addition of large filters [6].

In order to extract the important features from the output of the Convolution layers, we utilize a 1D MaxPooling layer. As depicted in Figure 2.1, this layer selects the maximum value in the provided pooling window. This allows the output of Convolution layers to be reduced in it's dimensions while preserving the important features learned.

2.2.4 Squeeze and Excitation Block

Hu et al. [25] proposed a squeeze-and-excitation block to be added to outputs of convolution blocks so that the inter-dependencies between channels are modelled by adjusting outputs of filters in two steps, squeeze and excitation. Here, the squeeze operation utilizes a global average pooling to obtain channel-wise statistics beyond the local receptive fields. This operation generates a channel-wise global average

over the temporal dimension T by calculating the statistics $z \in \mathbf{R^C}$ by shrinking the output of a convolution layer U through T [30]. Each element of z is calculated as:

$$z_c = \mathbf{F}_{sq}(u_c) = \frac{1}{T}\sum_{t=1}^{H}\sum^{T} t = 1u_c(t) \tag{2.8}$$

Information that were aggregated (squeezed) in this block are then passed through an excite operation to fully capture channel-wise dependencies. This operation learns nonlinear relationships between channels, and learns relationships that are not mutually exclusive over multiple channels. This is achieved by using a simple gate with a sigmoid activation:

$$s = F_{ex}(z, W) = \sigma(W_2\delta(W_1 z)), \tag{2.9}$$

where F_{ex} is a neural network, δ is a Rectified Linear Unit (ReLU) function, σ is the sigmoid function, $W_1 \in \mathbf{R}^{\frac{C}{r}\times C}$, and $W_2 \in \mathbf{R}^{\frac{C}{r}\times C}$ are learnable parameters, and r is the reduction ratio hyper-parameter [30]. Finally, this output is rescaled as:

$$\tilde{x_c} = \mathbf{F}_{scale}(u_c, s_c) = s_c \cdot u_c, \tag{2.10}$$

where $\tilde{X} = [\tilde{x}_1, \tilde{x}_2, \cdots, \tilde{x}_c]$ and $F_{scale}(\cdots)$ is the multiplication between feature map $u_c \in R^T$ and scale s_c. This squeeze and excitation block has been added to results of convolution layers by Karim et al. [30] to create a time-series classification model for multi-variate data. It allows for the models to learn global multi-channel dependencies from the outputs of convolution layers. This is important in multi-variate models as all feature maps learned in previous layers may not affect the subsequent layers with the same magnitude. With squeeze and excite, the model is conducting a learned self-attention on output features obtained from previous feature maps [30]. This model has

been shown to perform better than other forms of time-series-classification models. Therefore, this thesis also tests this new state-of-the-art time series classification model using data from ICS.

2.2.5 Anomaly Detection Framework

In this section, we discuss the attack scenarios to be detected by our proposed architecture, the proposed architecture for anomaly detection framework built utilizing the layers discussed in the previous section, as well as the metrics and loss function used.

Attack Scenario

The proposed methodology aims to detect data manipulation attacks that spoof and inject sensors/actuators values into the PLC. In this chapter, these data manipulation attacks are divided into two categories: (1) stealthy attacks and (2) non-stealthy attacks. Stealthy attacks slowly manipulate the sensors/actuators values to gradually cause the process behavior to deviate from the normal behavior. Impact of these attacks are not observed immediately or within a short time-window. Non-stealthy attacks manipulate the sensors/actuators value in shorter periods of time to force the system outside of normal cyclical behavior. Impact of these types of attacks can be observed through spikes or changes within the sensors/actuators values within a short time window.

In order to detect these two types of attacks, the detection model must learn the relationships between sensors and actuators over a short window as well as longer period of time. In ICS, this relationship dictates how change in one sensor/actuator affects other sensors/actuators over time. Therefore, we are learning the behavior of

these components without having to encode the rules of the PLC or physical process itself. This helps detect anomalies where change in one of the components does not trigger an expected change in another component. In the proposed architecture, Convolution layers are used in conjunction with LSTM layers to learn the relationships between features over different periods of time. As mentioned in section 2.2.3, Convolution layers can learn the temporal dependencies that can be used to identify normal sensors/actuators behavior, as well as known attack behaviors, in a defined time-window. Hence, they are useful at detecting non-stealthy attacks. At the same time, LSTM layers help identify normal as well as attack behavior for stealthy attacks due to their effectiveness at learning longer historical dependencies.

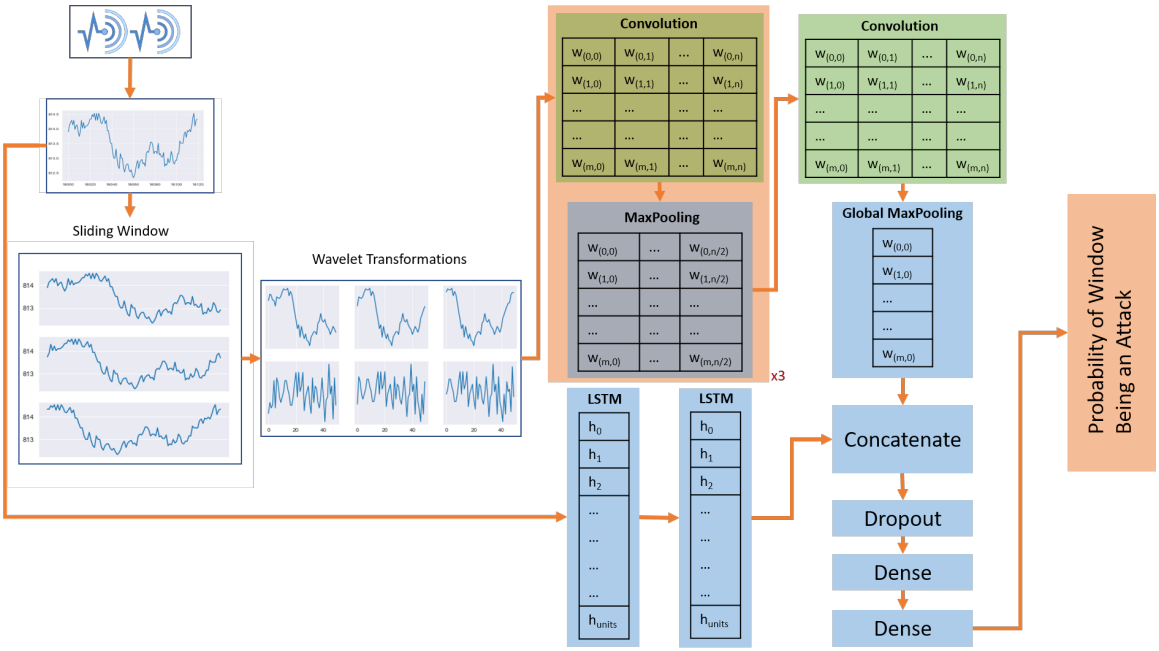


Figure 2.2: Classification Model using Convolutions, LSTM, and Dense Layers. Here, w, w', h, h', f, and f' represents the intermediate features.

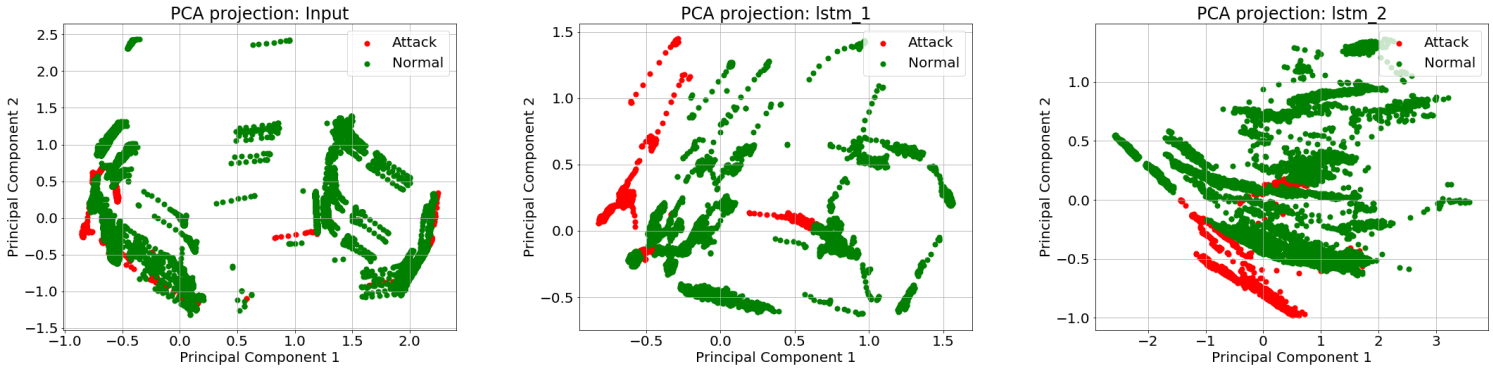
Non-stealthy Attacks Representation

As discussed in the previous section, Convolution layers will help facilitate anomaly detection of non-stealthy attacks within ICS. In the proposed architecture, Convolution layer accepts the sensors/actuators' value as the input features after the DWT transformation and learns how these features individually, and as a group, change over a time-window. This architecture contains a number of stacked convolution layers with the number of convolution layers treated as a hyper-parameter. The number of layers is dependent upon the data-set as well as efficiency of the model in learning the feature representation required for achieving better metrics. While our final architecture utilizes four convolution layers, this can be adjusted according to the results of a hyper-parameter search. Along with the number of layers, each Convolution layer requires hyper-parameters of $filter\_size$, $number\_of\_filters$, $dilation\_rates$, and $activation\_function$ to be defined. These hyper-parameters are picked based on empirical analysis of the results obtained from searching combinations of hyper-parameters values. Results of the hyper-parameter searches for each model is presented in their respective case study section 2.3.

As shown in Figure 2.2, each convolution layer is followed by a pooling layer. The first three convolution layers are followed by a 1D MaxPooling layer. In addition, a final Convolution layer is followed by a GlobalMaxPooling layer to reduce the dimension of data and pick the most important features from each Convolution filters.

Stealthy Attacks Representation

Along with CNN, LSTM layers are used to learn the long term temporal dependencies within the data beyond the specified time-window that CNN is being trained on.



(a) Original Features. (b) LSTM Layer 1 (c) LSTM Layer 2

Figure 2.3: PCA Decomposition of attack#23 and how it is represented after passing multiple LSTM layers of the architecture.

As mentioned in section 2.2.3, LSTM layers and their hidden cells can help learn the relationships between features in the input over a long period of time. With the help of the learned representation this architecture can identify attacks where sensors/actuators are targeted for large periods of time with gradual physical impact in the system. In this architecture, two LSTM layers are used to learn the long-term dependencies within the input data. Each LSTM layer is initialized with a hyper-parameter of $units$, where $units$ represents the number of units in cell hidden states whose activation's will be propagated to output. The initial LSTM layer returns full sequences of cell output (h_t) for every time-step fed into the layer, returning a three dimensional output of shape (batch size, t, $units$). This is then passed as input to the final layer and returns only the last output (h_t) sequence of shape (batch size, $units$). Output obtained from the LSTM layer will represent input features to effectively separate system normal behavior from anomalous behavior. These representation are visible in the Principal Component Analysis (PCA) decomposition, Figure 2.3, of the outputs from our two LSTM layers. As observed in Figure 2.3, initially the attack values are not separable from normal values within a data-set. But, moving this input through the trained models' LSTM layers helps obtain representation where they can be separated from each other.

In our architecture, LSTM cells contain 128 activation units, thus learning the long term dependencies and sending an output of (batch size, 128). Once the results from LSTM layers are received, they are concatenated with the results from the GlobalMaxPooling layer. This GlobalMaxPooling operation converts the outputs of the final Convolution layer to a fixed embedding in two dimensional space. This resulting feature representation is then passed through a dropout layer, with a dropout rate of 0.2, to regularize the learning through disabling random neurons activation.

Detection

As shown in Figure 2.2, results of the Dropout layer are passed through three fully connected Dense layers. These layers have units sizes of $[128, 64, 1]$. The final dense layer in the architecture outputs the probability ($[0, 1]$) of an input being an attack. Here, probability greater than 0.5 labels the time window as an attack and less than 0.5 labels the time window as normal. In order to achieve this probability bound, the final Dense layer uses a *sigmoid* activation function.

Optimizer and Loss Functions

Since our architecture is trained for a binary classification problem of identifying a time-window as normal or attack, we utilize the loss function of *Binary Cross-Entropy* with added class weights as the loss function during training. Binary Cross-Entropy is calculated by using equation (2.11), where $\mathcal{L}$ is the loss function, y is the true label, cw_1 is class weights assigned to attack label, cw_0 is the weight assigned to normal label, and $\hat{y}$ is the prediction from the model. Here, class weights are derived from ratio of occurrence of normal and attack classes within the training data.

$$\mathcal{L}(\hat{y}, y) = \frac{-1}{N} \sum_{i=1}^{N} [cw_1 * y_i \log(\hat{y}_i) + cw_0 * (1 - y_i) \log(1 - \hat{y}_i)] \tag{2.11}$$

Finally, we utilize *Adam* [32] algorithm as an optimizer during the model training process. *Adam* optimizer requires four parameters of α, β_1, β_2, and ϵ. We utilize the default values for the parameters proposed by Kingma and Ba [32], where $\alpha = 0.002$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1\text{e}{-07}$.

Metrics

The aim of our architecture is to detect anomalies and attacks within the process of ICS based on the sensor and actuators data. To measure the performance of this architecture, we focus on the metrics of precision (equation 2.13), recall (equation 2.12), F1-score (equation 2.14), and numbers of attacks detected. Recall is the measure of a models ability to detect true positives within the given data. Hence, it is a useful metric for selecting models for a domain with larger cost associated with false negatives. Precision is the ratio of correct attack detection by model to all alarms raised by the model. This metric is useful in selecting models for a domain where large number of false positives can be costly. Finally, F1-score strikes a balance between the precision and recall metrics. It is an important metric for anomaly detection models in ICS where costs between false negatives and false positives must be balanced. We expect our final model to have a higher F1-score, while maintaining a higher percentage of attacks detected. Therefore, along with the measurement of the previous three metrics, we will also calculate recall for each individual attack window. This is carried out to verify the performance of the model within each attack period. For the purpose of this study, an attack is defined to be detected if any time-window containing the attack point is detected at least once.

$$recall = \frac{tp}{tp + fn} \tag{2.12}$$

$$precision = \frac{tp}{tp + fp} \tag{2.13}$$

$$F1 - score = 2 \times \frac{precision \times recall}{precision + recall} \tag{2.14}$$

where, tp = number of windows correctly identified as attack, fn = number of windows incorrectly identified as normal, and fp = number of windows incorrectly identified as attack.

2.2.6 Hyper-parameter Search

In deep learning, hyper-parameters are a set of parameters that can be controlled to vary the learning process. These parameters are not learned by the models and must be set during definition of the models to be trained. In the anomaly detection model discussed above, hyper-parameters of kernel size, activation units, batch size, dilation rates, window size, and filter size need to be optimized to the specific dataset being used. Within this thesis, an automated grid search strategy is utilized to find the optimal value for these hyper-parameters. Grid search is an exhaustive search within a manually defined subset of values for the hyper-parameters. This type of search requires the user to define a performance metric, and a discretized set of values for each hyper-parameter. For this anomaly detection model, F1-Score (Equation 2.14) on a validation split of original data is set as the performance metric. Sets of values are defined for each hyper-parameter to lower the search space. Since these sets are dependent on the dataset being trained, they will be presented in section 2.3.

2.2.7 Implementation

Algorithm (1) summarizes the implementation of our methodology for training the anomaly detection model described in the sections above. Sensors and actuators data from ICS (X), indexes of sensors in the data-set (S), manually encoded attack labels (Y), batch size (m), and window size (ws) are passed as input to train the model. Initially, sensors and actuators data are scaled to a magnitude of $[0, 1]$ based

Input : Time-series data $X = \{X_i\}_{i=1}^{n}$, S sensors index in X, $Y = \{Y_i\}_{i=1}^{n}$ attack labels, batch size m, window size ws, training samples n
Output: Trained weights for neural network model
$X\_transformed \leftarrow MinMaxTransform(X, range = (0, 1))$
forall $j \in \{1, 2, \cdots, (n - ws)\}$ **do**
  $X\_ws_j \leftarrow X\_transformed_j^{j+ws}$
  if $Y_j^{j+ws} == 1$ **then**
    $y\_ws \leftarrow 1$
  else
    $y\_ws \leftarrow 0$
  end
end
$X\_transformed \leftarrow DWT(X\_ws[S]) \oplus X\_ws[!S]$
$X\_train \leftarrow X\_ws_1^{0.4\times n}$
$X\_test \leftarrow X\_ws_{0.4\times n}^{n-ws}$
$y\_train, y\_test \leftarrow y\_ws_1^{0.4\times n}, y\_ws_{0.4\times n}^{n-ws}$
Randomly initialize internal weights of model w.
for *each iteration* **do**
  forall $k \in \{1, 2, \cdots, \frac{0.4\times n}{m}\}$ **do**
    Sample a batch X_train_k from X_train;
    Split batch into $[X_k^1, X_k^2, \cdots, X_k^l]$;
    Feed batch into CNN+LSTM and get two output sequences $[h_{ck}^1, h_{ck}^2, \cdots, h_{ck}^l]$, and $[h_{pk}^1, h_{pk}^2, \cdots, h_{pk}^l]$;
    Concatenate sequences together, flatten into vector O, and dropout 20%;
    Pass O through Dense layers and final layer ending in *sigmoid* activation function to obtain probability of input being attack;
    Update internal weights w by using weighted binary cross-entropy (equation 2.11) and *Adam* optimizer;
  end
end

Algorithm 1: Summary of Methodology.

on the distribution of a small subset of the data. Following the scaling, the data is transformed into a sliding window format using the hyper-parameter ws. Each window of data goes through a Discrete Wavelet Transformation on all the sensors data. The transformed data are appended to the input along with the original actuators data. Resulting data is then split into train-test split for training of the model. Here, a low percentage of data is used for training to get a performance metric of model on new attack behaviors. After the data is obtained, a neural network model is trained using a batch size of m with the process described in section 2.2.5. This training algorithm uses a weighted binary cross-entropy function to update the internal weights of the neural network. The obtained trained weights are then used to test the anomaly detection model against data it did not observe during training process to provide the complete result of the anomaly detection framework. This model can then be deployed to catch attacks in real-time within the processes it is trained on.

2.3 Case Study

The effectiveness of the proposed framework is verified utilizing the existing sensors and actuators' data from the Secure Water Treatment (SWaT) [35] test-bed. SWaT is a functional miniature replica of a water treatment plant designed by the Singapore Institute of Technology [22]. Figure 2.4 provides an overview of the water treatment system, where the treatment process is divided into six sub-processes (denoted as P1 to P6) that produce 5 gallons/minute of doubly filtered water. Each of these processes are controlled by a primary Programmable Logic Controller (PLC), and have a redundant hot-standby PLC. The water treatment process starts at P1, as

incoming water is taken in and stored in a tank. Then, the water is passed to P2 for quality assessment, and chemical dosing is carried out to bring the water quality to an acceptable threshold. In P3, the water is led through a fine set of filtration membranes to remove the undesired materials and in P4 the chlorination and de-chlorination process is conducted. Then, the water is pumped to P5 for the Reverse Osmosis (RO) process. Finally, in P6a and P6b, some of the water is stored to carry out backwash, and recycled mimicking real-world treatment plants where water is flown out of the plant for distribution.

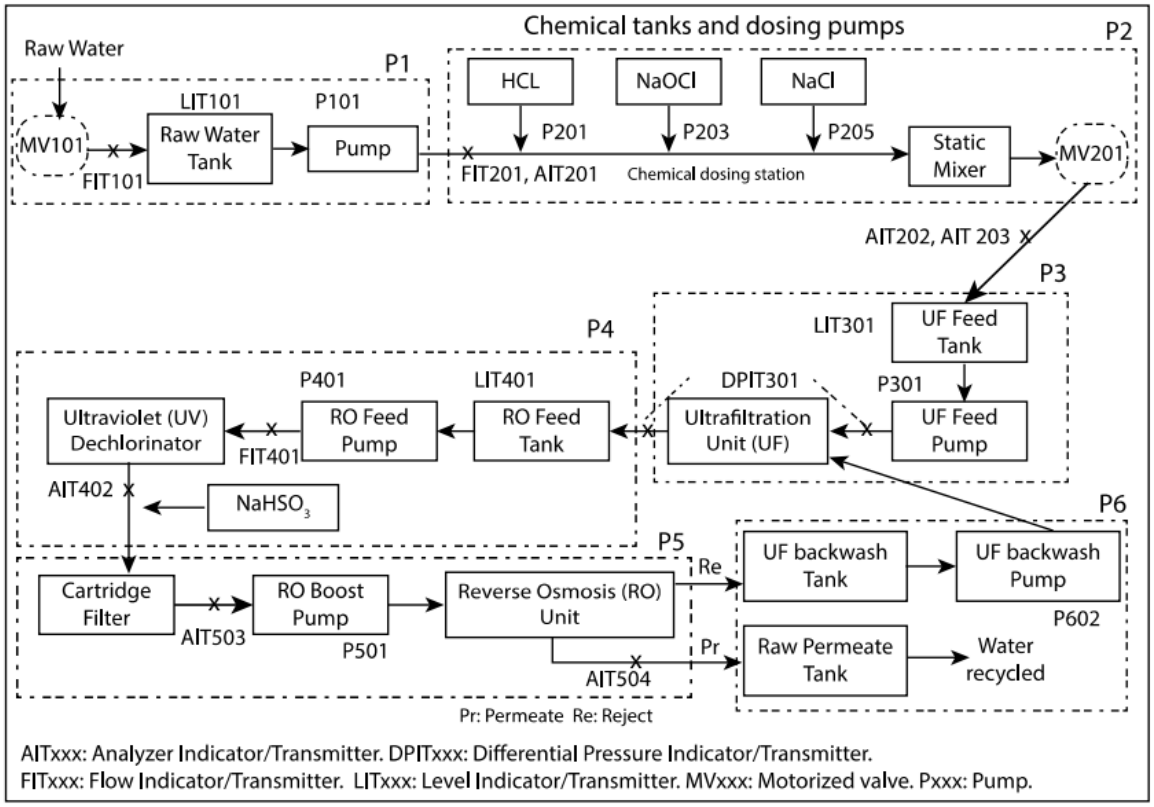


Figure 2.4: Overview of the SWaT testbed showing the processes segmentation along with the location of sensors and actuators within the system [39].

2.3.1 SWaT Data-Set

SWaT data-set [22] contains over 11 days of continuous operational data of 51 sensors and actuators (Table 2.1) that are part of the six water treatment processes. The data is collected over seven days of normal operation of the system, and four days of hackathon event where a total of thirty-six successful attacks were carried out. The description of the attack scenarios can be found on the website [2] [22]. Most of the attacks are based on the injection of process variables values into the Programmable Logic Controller (PLC) and resulting in the PLC accepting the spoofed value as real sensor data. There are some stealthy attacks where the sensor values are modified slowly within the respected process. Overall, the data-set contains 53 columns (1 for timestamp, 1 for attack label, 51 for sensor/actuator values) and $912,791$ rows where each row is a value recorded per second.

Table 2.1: Sensors and Actuators in SWaT Data

Process	**Sensors**	**Actuators**
1	FIT101, LIT101	MV101, P101, P102
2	FIT201, AIT201, AIT202, AIT203	MV201, P201, P202, P203, P204, P205, P206
3	FIT301, DPIT301, LIT301	MV301, MV302, MV303, MV304, P301, P302
4	FIT401, AIT401, AIT402, LIT401	UV401, P501, P402, P403, P404
5	FIT501, FIT502, FIT503, FIT504, AIT501, AIT502, AIT503, AIT504, PIT501, PIT502, PIT503	P501, P502
6	FIT601	P601, P602, P603

[2] https://itrust.sutd.edu.sg/itrust-labs_data-sets/data-set_info/#swat

2.3.2 Data Pre-processing

In order to deal with the high cardinality and noisy data in SWaT, the following steps are performed:

Scaling and normalization

In order for each feature to contribute effectively in the learning algorithm's output, the features are scaled to a range of $[0, 1]$. In this case, the normalization of the features are not feasible because the sensors measurements and actuators' status do not follow a normal distribution. Therefore, normalization of the features would result in the loss of information about their original distributions.

Slicing and window based segmentation

A fixed size sliding window (ws) is selected as a hyper-parameter. In the data-set of i entries, total number of $(i - ws + 1)$ windows are required. In order to determine ws, in this research a hyper-parameter search is carried out in the range of $50 - 300$ that resulted in the window size of 200 as the best fit for each model.

Performing wavelet transformation

Once the data is split into time windows, wavelet transformation is carried out on the sensor data for each time window. In the SWaT data-set, after this transformation the number of features are increased from 51 to 76 features.

Finally, the data is divided into a train/test sets by having 40% of the data as a training set and the rest as a testing set. In this training set, 0.06% of the data is labelled as an attack data. The reason for including the attack data as part of the

training set is to increase the effectiveness of the detection algorithm. Therefore, the model is trained on a small proportion of the attack data, while the remaining attack data is included in the testing set to validate the performance of the model against new attacks.

2.3.3 Hardware Specification

Training is carried out in the Google Compute cloud instances. Different configurations of machines are picked based on the type of model we are training and the hardware that need to be used for them. The Convolution only models are trained in a cloud compute instance with NVIDIA Tesla K-80 GPU (12 GB RAM) and 61 GB of memory. Whereas the CNN+LSTM models are trained with a cloud compute instance with 8 NVIDIA Tesla V100 GPU (32 GB RAM) and 80 GB of memory. To train the SVM model, we utilized a processor optimized computing instance with 8 vCPU and 64 GB RAM.

2.3.4 Anomaly Detection Results

This section presents the results of different architecture of anomaly detection models based on the SWaT data-set.

SVM Results

A baseline study is performed based on the Support Vector Machine (SVM) method as presented by Inoue et al. [27], to verify the performance of unsupervised methodologies in this data-set. One class classification SVM using Radial Basis Function (RBF) kernel is used to learn a classification boundary for normal data (not including the attack information) and identify the outliers. We used Radial Basis kernel Function

Recall	Precision	F1-score
0.699	0.925	0.796

Table 2.2: Results of one-class SVM model.

(RBF) to estimate the similarity between two sample data and transform the data. This Kernel is defined as:

$$K(x, x') = exp(-\gamma \left\| x - x' \right\|^2), \gamma > 0 \tag{2.15}$$

where the x and x' are two sample data and γ is the kernel parameter. The kernel map $\phi : X \to H$ is a function that transforms a feature vector from the original space X to a new one H.

Once the one class-classification model is trained, it is tested on the attack data to detect anomalies that are not within the learned classification boundary. Results of the SVM (Table 2.2) show that the model performed with high precision, however it was not able to detect 16 attacks and took a long time to train (4 hours). This inefficiency and failure to detect anomalies were also reported by other researchers who utilized unsupervised methodologies [23, 27, 42] on SwaT dataset. Therefore, this research is focused on supervised deep learning methodologies.

CNN

The initial experiment is based only on the Convolution-Maxpooling layers where the ability of CNN to learn temporal dependencies within a time-window helps identify normal and attack behaviors.

As mentioned in section 2.2.5, CNN is able to learn the relationships between sensors/actuators within a time window to detect non-stealthy attacks. For example,

learning the relationship between Level Indicator (LIT) 101, Motor Valve (MV) 101, and Flow Indicator (FIT) 101, allows the model to infer the presence of anomalous behavior within the attack window. Therefore, an attack that results in tank overflow in Process 1 through the manipulation of LIT101 and MV101 (Figure 2.5) can be detected by the model.

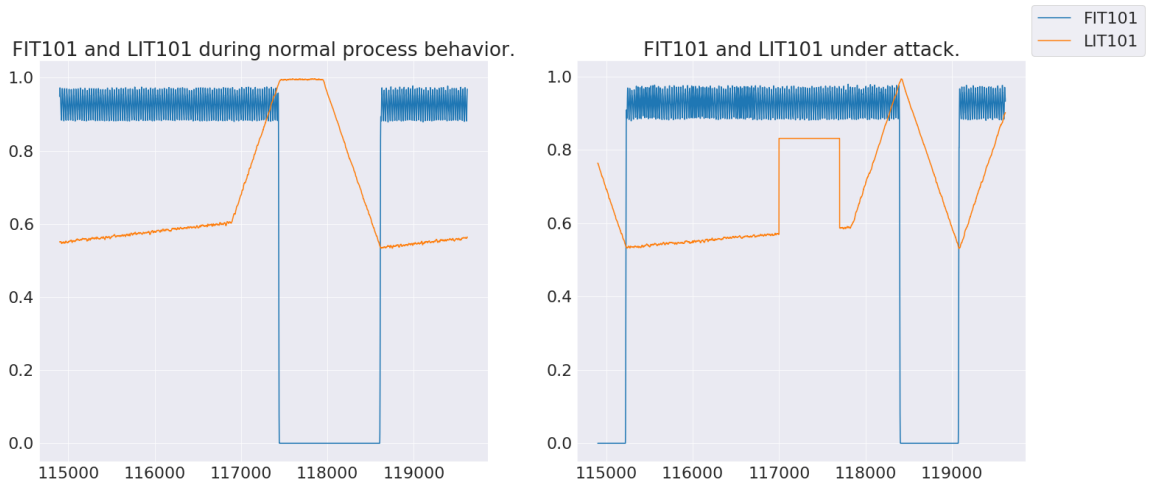


Figure 2.5: Relationship between LIT101 and FIT101 (Left) under normal behavior, (Right) under non-stealthy attack

Hyper-parameters Hyper-parameter search space for this model is limited to the parameters of dilation rate, window size, and filter size. A grid search for these hyper-parameters resulted in the filter size of 5, number of filters of $[2, 4, 8, 16]$ for each layer of convolution, dilation factors of $[2, 4, 8, 16]$, batch size of 100, and window size of 200 as the best fit for this CNN architecture.

Results Training time for this model was 30 minutes and cost of training was approximately \$1.25 when computed in a cloud instance. Metrics of this model are summarized in Table 2.3. While the metrics show performance higher than previous

works, this model still fails to catch 13 attacks in the data. Therefore, we add LSTM layers to our model in order to learn feature representations for stealthy attacks.

Recall	**Precision**	**F1-score**
0.69	0.58	0.65

Table 2.3: Results of our supervised classification model using CNN.

CNN + LSTM

The success of CNN provided us with groundwork to try and improve the results and catch attacks that were not caught in previous architecture. As described in Section 2.2, Convolutions can learn time windowed relationships, but they do not offer long-term sequence learning. In order to incorporate the goal of long-term sequence memorization, we add LSTM cells in our experiments. Hence, the architecture discussed in Section 2.2.5 is trained to created an anomaly classification model. In this architecture, usage of LSTM can help classify behaviors that gradually change values of sensors/actuators to cause damage to the physical process. For example, attack number 3 (Figure 2.6) in the testbed slowly increases the value of LIT 101 by $\frac{1mm}{sec}$ to cause tank underflow and pump damage. These changes that do not follow regular pattern of change over a gradually increasing time period can be represented by the LSTM layer.

Initially, this model was trained with 1 LSTM layer, but the model had a low recall score improvement from the model without LSTM layers. After increasing the number of LSTM layers to 2, we increased the recall score and numbers of attacks detected. These results and the hyper-parameters used to obtain them are discussed next.

Hyper-parameters In order to find the best model, there are multiple hyper-parameters that need to be optimized for in our model. Namely window size, batch size, kernel size of Convolutions, and dilation rates need to be picked to improve the performance of the model. To this end, we explore the space of $[1, 10]$ for kernel size, $[0.2, 0.3, 0.4]$ for dropout percentage, filter sizes are changed to match the output size of the LSTM output, and LSTM activation units are in the set of $[16, 32, 64, 128]$. Window sizes were searched in the space of $[50, 300]$ with 50 steps increment, and dilation rates were searched in the space of $[2, 8]$ where dilation on each subsequent layer was a multiple of 2^n with n being the layer number of Convolution starting with first Convolution layer. We utilize a grid-style search for the hyper-parameter, and aim to optimize for the F1-score. Initially, hyper-parameter search is capped to 20% of the search space to study the importance of these parameters on the testing metric. Once the initial results of the grid-search is saved, a RandomForest Regression is used to study the importance of changing hyper-parameters in the F1-score of resulting model. Figure 2.7 shows that change in dilation factor, and number of lstm units have

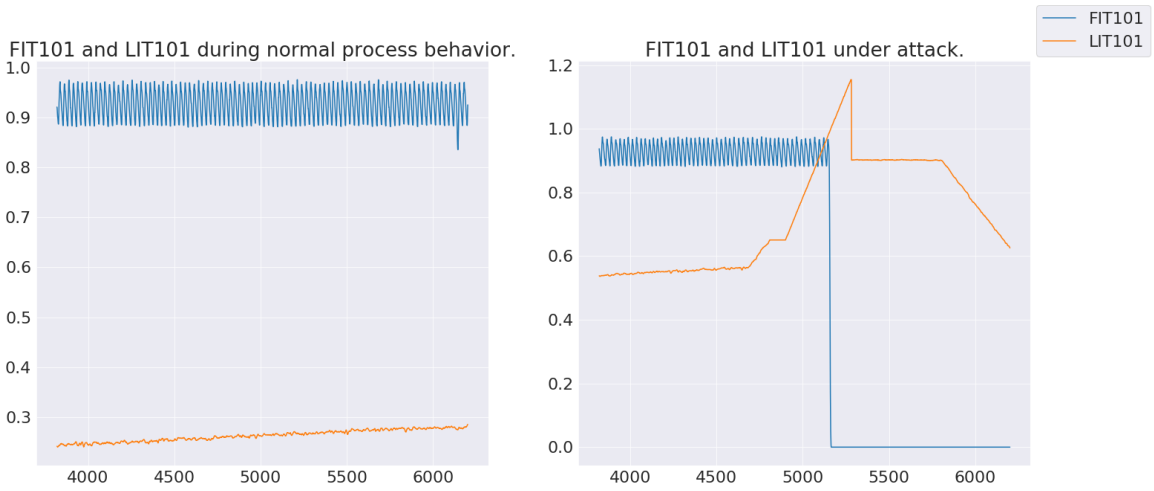


Figure 2.6: (Left) Under normal behavior. (Right) Example of Stealthy attack on LIT-101

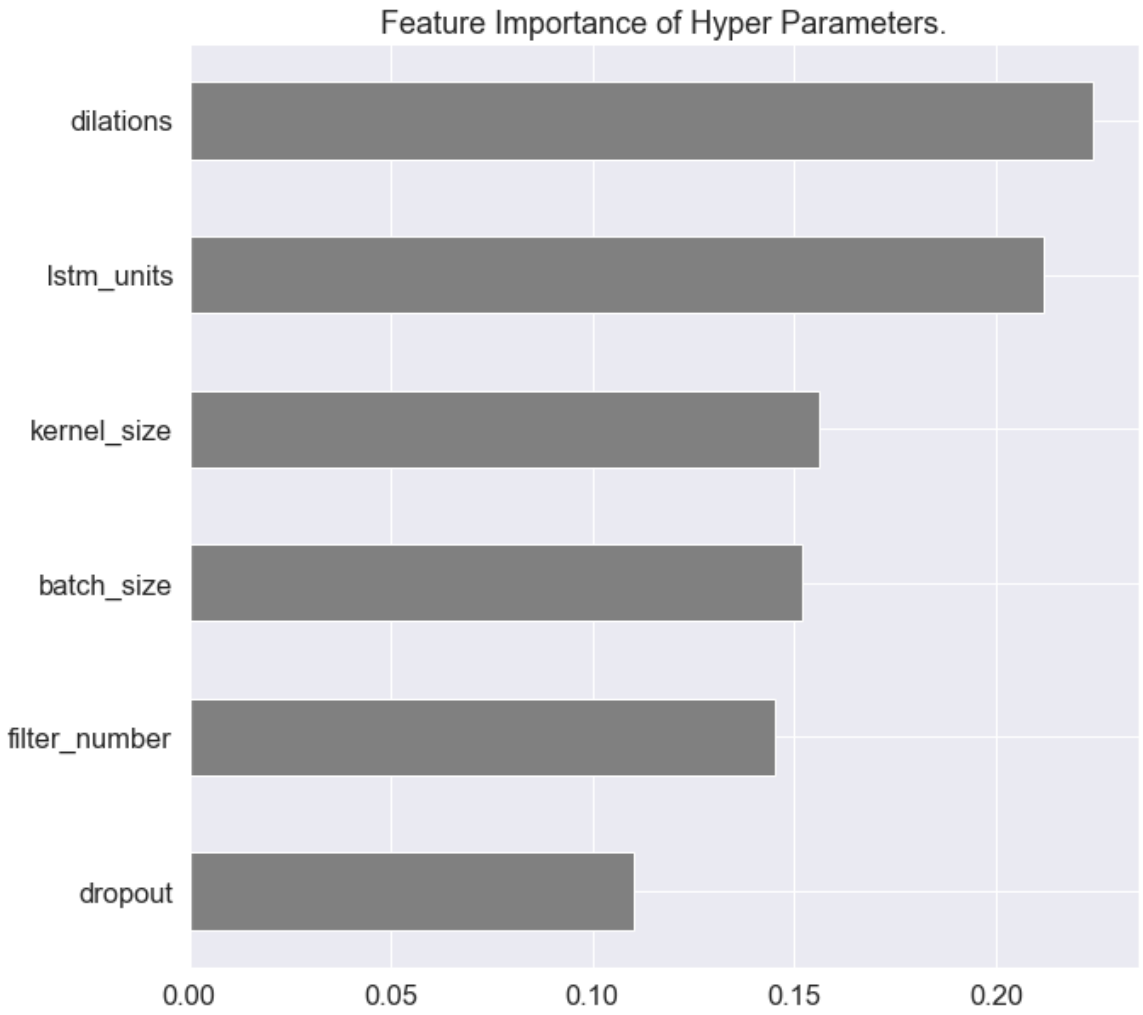


Figure 2.7: Relative Importance of hyper-parameters in F1-Score of model.

the largest impact on the F1-score of the model. Finally, a hyper-parameter search is run on the entire space for these two hyper-parameters and 50% of the search space for the remaining hyper-parameters.

Following the results of the search, the anomaly detection model is trained with a batch size of 200 for 24 epochs. The values for our hyper-parameters were a Window size of $200 seconds$, kernel size of 4, number of filters of $[5, 10, 15, 20]$, and dilation rates of $[2, 4, 8, 16]$. Number of LSTM units is selected to be 128 units. Epoch size

was not a part of our hyper-parameter search. Smaller epoch size of 24 is used as a training stop as the model converges to a stable loss around 24 epoch (Figure 2.8) while obtaining the lowest loss values out of all models explored.

Results Results of this supervised classification model is shown in Table 2.4. The final model, with 76 features for each window, finished training in 25 minutes while only missing 9 attacks. This level of attack detection is the best performance by a supervised as well as unsupervised [27] methodology in this domain. As shown in Table 2.6, per attack recall rates of our model performs better than the results in Inoue et al. [27]. At the same time, our model is trained in 25 minutes at a cost of approximately $2.00 which is significantly lower than 2 weeks that it took for the previous DNN methodology to train.

Recall	**Precision**	**F1-score**
0.773	0.972	0.861

Table 2.4: Results of our supervised classification model using CNN+LSTM with Wavelet Transformation.

While individual recall rates as well as detection times are shown in table 2.6, Figure 2.9 presents an example of detection and detection time for attack #24. In this attack, an attacker sets LIT-401 to values lower than the lowest acceptable threshold in order to cause a tank overflow. As shown in the figure, our first alarm is raised 44 seconds after the attack begins, and 94% of the attack window is correctly labelled as an attack. This helps detect the attack before it causes physical harm to the pumps in the system. Similar detection patterns were observable for most attacks detected by our model. Further discussions on the attacks this model failed to detect are done in section 2.4.

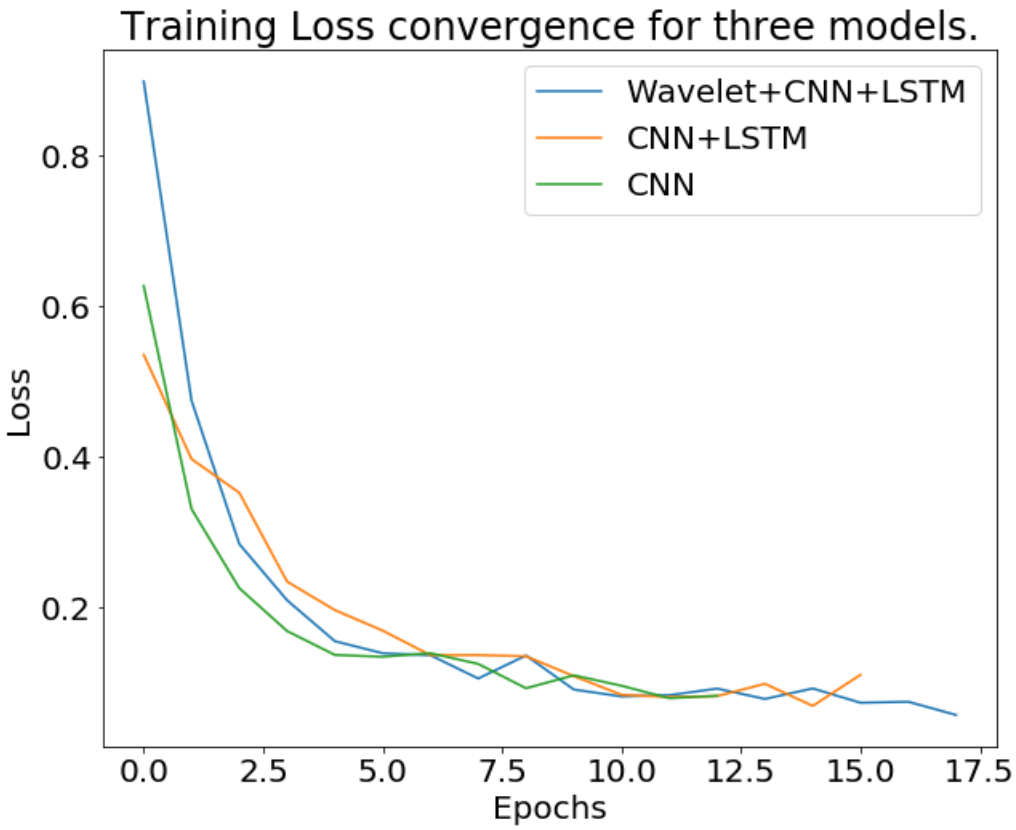


Figure 2.8: Training loss for three trained models. CNN+LSTM loss converges at lower value at 24 epochs.

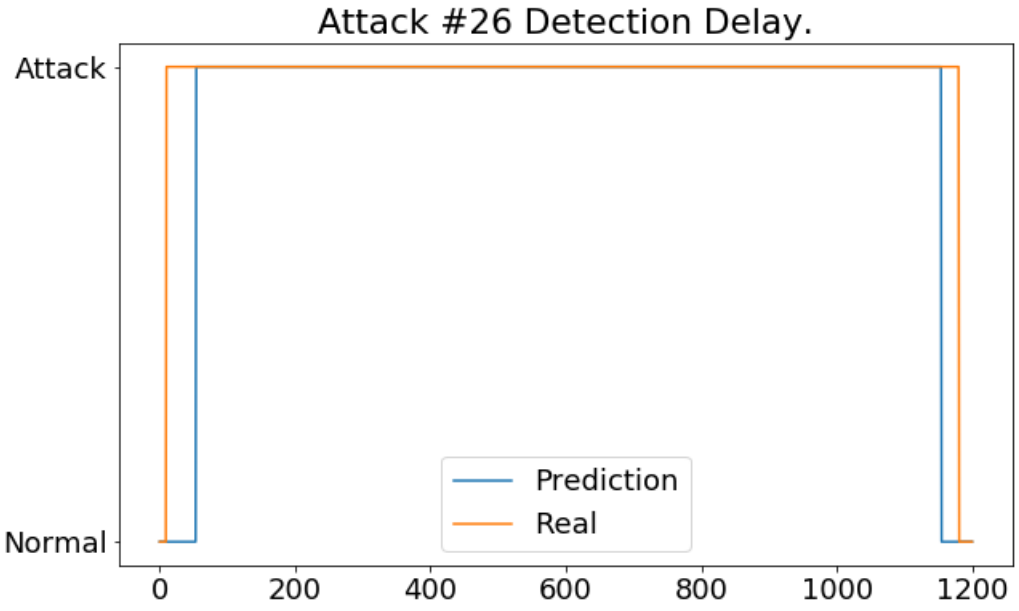


Figure 2.9: Example of attack detection by our model for attack number 26 (table 2.6).

2.3.5 Squeeze-and-excite CNN+LSTM

The CNN+LSTM model defined above set the state-of-the-art results within SWaT dataset. In order to verify the performance of this new architecture, it is compared to a state-of-the-art time-series classification architecture defined by Karim et al. [30]. As shown in figure 2.10, this model utilizes two blocks of 1D Convolution + Batch Normalization + Squeeze-and-excite layers followed by a 1D Convolution + Global-MaxPooling block. At the same time, the input is fed through a LSTM + Dropout layer. Results of both layers are then concatenated and passed through a softmax layer to obtain the probability of input being an attack.

Hyper-parameters For our final model, the LSTM layers have 128 activation units, and the dropout layer has a drop percentage of 20%. Convolution layers have

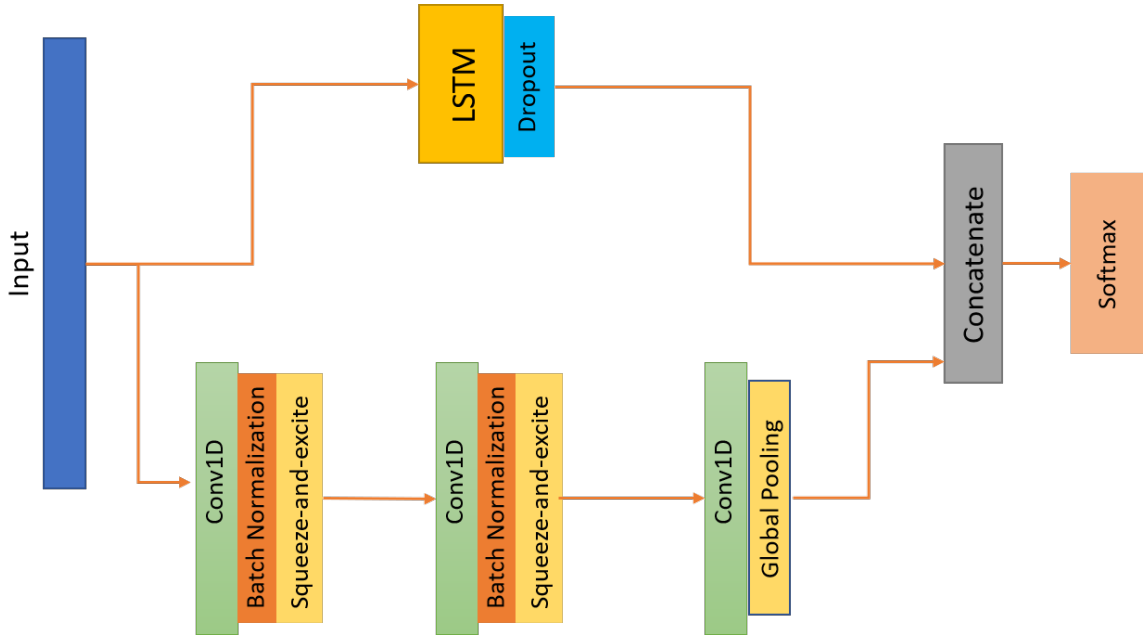


Figure 2.10: Squeeze-and-excite archtiecture variant defined by Karim et al. [30]

filter sizes of $[10, 20]$, kernel size of 6, two L2 kernel regularizers, and dilation rates of $[3, 9]$. The squeeze-and-excite layers have a reduction ratio of 2.

Result Results of this state-of-the-art time series classification model are presented in Table 2.5. This attack detection model could successfully detect 26 out of 36 attacks, successfully performing better than SVM, and CNN architectures, but performing worse than a CNN+LSTM model by 1 attack. This was as expected, as this is a recreation of a state-of-the-art time-series classification model. But, this model still had lower F1-score, recall, and precision, than the architecture built on this thesis.

Recall	**Precision**	**F1-score**
0.62	0.85	0.72

Table 2.5: Results of squeeze-and-excite architecture variant

Table 2.6: Recall rates and detection delay for each attack is compared to baseline formed in Inoue et al. [27] and Shalyga et al. [42] Detection Delays of N/D mean attacks were not detected, while N/A signifies unavailability of time in baseline paper.

Attack #	Attack Points	Recall (Our method)	Recall (Inoue)	Detection Delay (Our method) in *seconds*	Detection Delay (Shalyga) in *seconds*
1	MV-101	1	0	83	N/D
2	P-102	1	0	40	112
3	LIT-101	1	0	11	N/D
4	MV-504	1	0	41	N/D
5	AIT-202	1	0.717	00	11
6	LIT-301	1	0	26	42
7	DPIT-301	1	0.927	80	16
8	FIT-401	1	1	5	18
9	FIT-401	1	0.978	5	3
10	MV-304	1	0	1	N/D
11	MV-303	1	0	0	N/D
12	LIT-301	1	0	2	297
13	MV-303	1	0	29	N/D
14	AIT-504	1	0.123	4	13
15	AIT-504	1	0.845	26	N/D
16	MV101, LIT-101	1	0	0	10
17	UV-401, AIT-502, P-501	1	0.998	200	16
18	DPIT-301, MV-302, P-602	1	0.876	2	102
19	Turn off P-203 and P-205	1	0	174	53
20	LIT-401, P-402	0	0	N/D	1043
21	P-101, LIT-301	0	0	N/D	67
22	P-302, LIT-401	1	0	25	N/D
23	P-302	1	0.936	0	1164
24	P-201, P-203, P-205	0	0	N/D	52
25	P-101, MV-101, LIT-101	0	0	N/D	105
26	LIT-401	1	0	44	102
27	LIT-301	0	0	N/D	N/A
28	LIT-101	0	0	N/D	N/A
29	P-101	1	0	253	89
30	P-101, P-102	0	0	N/D	82
31	LIT-101	0	0	N/D	110
32	P-501, FIT-502	0	1	N/D	79
33	AIT-402, AIT-502	1	0.923	60	73
34	FIT-401, AIT-502	1	0.94	61	71
35	FIT-401	1	0.933	0	71
36	LIT-301	1	0	1021	N/D

2.4 Discussion

This chapter proposed a CNN+LSTM based supervised deep learning anomaly detection model for detecting stealthy and non-stealthy attacks targeting sensors/actuators in Industrial Control Systems (ICS). As mentioned in section 2.1, previous criticism of supervised anomaly detection methodology within this domain have focused on their inability to catch novel attacks that were not available during training of the model. Analysis of the final model explored in this chapter shows that supervised models can empirically perform better than semi-supervised models in catching known as well as unknown attacks (Table 2.6). This shows the ability of supervised methodologies to be similarly effective to unsupervised methodologies in the context of anomaly detection within the domain of ICS. This architecture also performed better than a state-of-the-art time-series classification model utilizing squeeze-and-excite blocks.

Along with criticism of supervised methodologies, there have also been criticism of deep learning methodologies and their inability to expand to an entire testbed [23], and maintain an inexpensive training cost [27]. In this chapter, we provide a training architecture with feature engineering that can train on the entire data of a water treatment plant, while maintaining low cost (Figure 2.11) and high detection rate (Table 2.6). With the availability of high power computing at low hourly costs, industrial sectors can add a layer of security to their processes by leveraging deep learning methodologies and creating an anomaly detection system with low cost and time penalty.

While our model achieves better performances than previous deep learning methodologies, it still fails on detecting 9 attacks within the SWaT data-set. Multi-point attacks [20, 21, 24, 25, 30, 32] and three single point attacks [27, 28, 31] are not detected

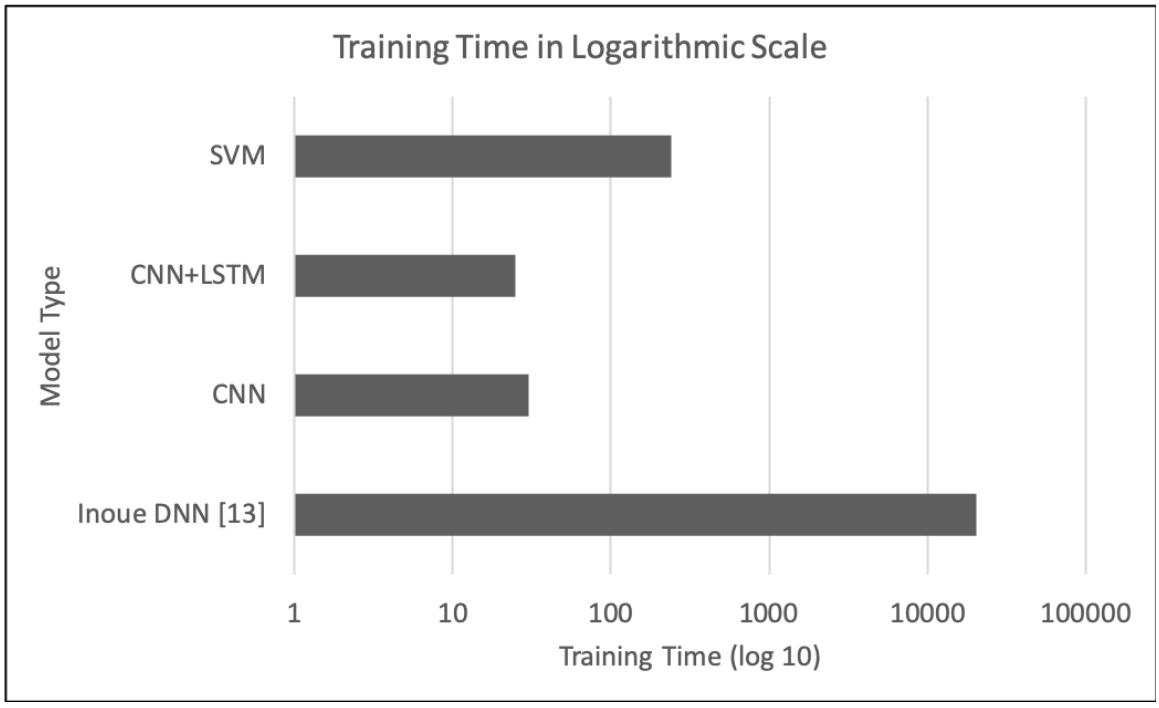


Figure 2.11: Training Time of our models compared to other state of the art deep learning model.

by our anomaly detection model. Within these attacks, attack numbers $[27, 28, 31, 32]$ did not cause actual change within the system and failed in their goal of violating the safety properties of the system. Therefore, the failure to detect these attacks are minimal within the SWaT testbed. Attack number 30 keeps the pumps P-101, and P-102 off to stop water outflow into the other processes. While this attack succeeds, the attack does not cause the other processes to be damaged in the time it took for the attack to start and stop. Therefore, 4 of the 9 undetected attacks cause physical harm to the system. In attack 20 and 21, level indicator value is set to significantly small number above threshold while attacking another component at the same time. These attacks were harder for our model to catch as there are noise in level indicator training data where values fluctuate despite needing to be a single value. Further exploration of reducing noise on sensor data can help in reducing these missed detections.

CHAPTER 3

EFFECTIVENESS OF ARCHITECTURE

The previous chapter produced a methodology for developing an anomaly detection model for sensors/actuators data obtained from ICS. This methodology was built with the premise of utilizing all sensors/actuators data available to create a single monolithic model for the system. This anomaly detection model could have been built in a distributed fashion, with one anomaly detection model trained on a single process. While the effectiveness of the final model was empirically established, the effectiveness of the model when trained on a single process compared to all processes at once was left unexplored. This chapter provides an analysis of the effectiveness of our anomaly detection model when it is trained on all processes at once, and when multiple models are created with each model representing a single process.

3.1 Related Works

Architecture for training and deployment of models within a large ICS is an important research question. While the idea of distributed training and deployment are discussed in the literature, the empirical effectiveness of different deployment patterns is not explored.

In their assessment of anomaly detection systems in ICS, Adepu et al., [2] introduced an open research question of the effectiveness of architectures and model of

deployment for anomaly detection system in ICS. After monitoring anomaly detection systems during a hackfest, they came to the conclusion that the current generation of detection systems are not enough to stop cyber-physical attacks. They also discuss the lack of analysis on how to train detection systems, and how to distribute them within a large process.

The open question is followed by Adepu et al., [3] with a method of placing invariant detection mechanisms in each controller to carry distributed attack detection. Here, invariants were generated based on the conditions set for the code base of each controller. One type of invariant is generated using State Conditional Graphs (SCG), where they are used to represent conditions that must be met for components controlled by a single controller. SCG are then used as state dependent invariants to be placed within each controller. Another type of invariant generation is also used where State agnostic invariants are built to check when average change in a system from time t to $t + 1$ increases beyond specified threshold. These invariants are then distributed to individual controllers throughout the system.

Although this methodology introduces ideas of distributed model training and deployment in ICS, it lacked empirical evidence supporting distributed architecture, and failed to answer the problem created by resource competition between process control code and invariant checking code in a single controller.

Following the previous idea of distributed invariants checking, Umer et al. [43] introduced the idea of distributed learning of invariants in ICS and deploying models in a distributed fashion. They use Association Rule Mining (ARM) on time-sampled data to build invariants within systems. First, Frequent item-sets are calculated on selected attributes within the data. In their case study, they use SWaT data where fifteen attributes are selected from the fifty one available. This reduced item-set is

then fed to a learning algorithm to generate association rules and confidence levels. Rules are then distributed to individual controllers if they only contain components within the controller, and to multiple controllers if they contain multiple components from different controllers.

Similar to the previous work carried out, this distributed training and deployment of the model is not empirically tested to provide evidence for a specific model over another. At the same time, Item-sets are generated by converting continuous features to discrete values values, losing information on the way. Furthermore, attributes are manually selected and reduced to avoid large computation overload at the cost of losing information.

Limitations This section described previous proposals of monolithic and distributed architectures usage in ICS. The effectiveness of these architectures are limited to anecdotal explanation and this is not enough to determine the correlations between effectiveness and architecture used. In this research, we aim to answer our *RQ 2* to empirically measure the effectiveness of different architectures of the anomaly detection model. This will allow us to find correlations between effectiveness, in terms of the metrics explored in *RQ 2*, and model architecture used.

3.2 Methodology

Within ICS, processes are assumed to be physically dependent upon each other and their environment. These dependencies between components can be an important information to learn in order to build an anomaly detection module. At the same time, ICS are built in modules, where independent controllers control different sub-processes. This architecture shows that normal states can be inferred

from individual sub-processes without the need to observe the entire process as a whole. These arguments can be directly translated to training and deployment of anomaly detection detection models within this domain. An individual model trained on all processes could capture the global relationships between components, while smaller models trained on a single process could learn the dependencies observed by a single controller. Therefore, this chapter explores two methods of training anomaly detection models and determines their effectiveness:

Monolithic Model In order to capture the overall dependencies of an entire system, we explore training our anomaly detection model using all the features available for the system to build a monolithic model. This model must be deployed with hot backups, such that we do not create a single point of failure.

Distributed Model To match the design of ICS controlled processes (Figure 2.4), anomaly detection model is trained and deployed in a distributed fashion. Each model is trained and observes features that are used by a single controller used to control a single process.

3.3 Results

Monolithic and distributed anomaly detection versions of the supervised anomaly detection architecture discussed in Chapter 2 are trained on the hardware described in Section 2.3.3. These models are trained four times, and their average results are presented in this section.

Findings of these experiments show a significant number of attacks caught by the the individual models (Figure 3.1). In combination, the Convolution+LSTM models

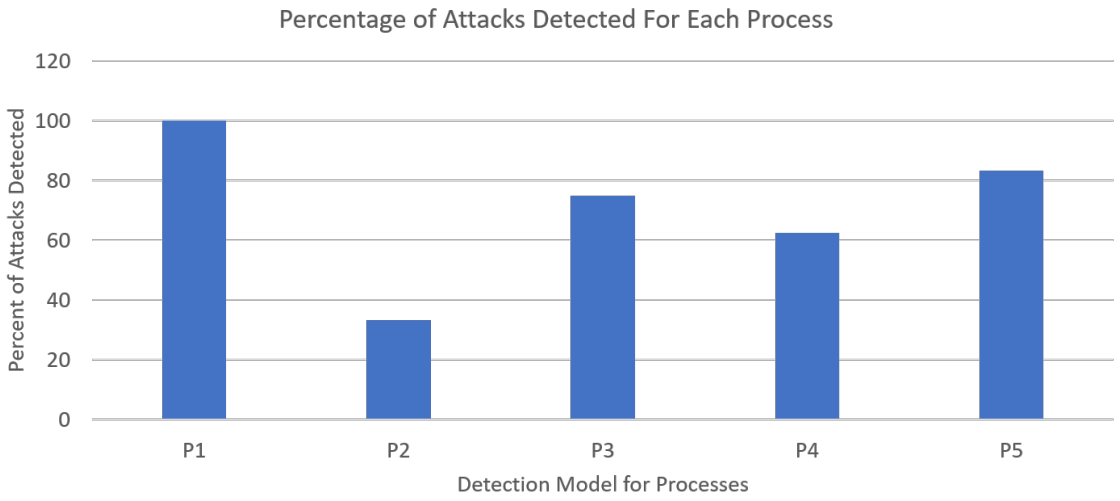


Figure 3.1: Attack Detection rate of distributed models trained on individual processes.

caught an average of 25 out of 36 attacks within the data-set, and the Convolution only models caught an average of 23 attacks. Although both of the individual models detected these attacks, two processes ($P4$ and $P5$) displayed significantly low F1-score (Figure 3.2) in all runs. These low scores highlight the struggle of these models in carrying out true classifications, leading to multiple false positives and false negatives. This could be attributed to the models failing to learn a global relationships that can aid in detecting some attacks.

On average, the monolithic models performed better than the distributed models. Compared to distributed models, the monolithic models were trained on all processes in a significantly lower amount of time (Figure 3.3) and achieved a higher F1-score. Following these results, it can be concluded that monolithic models should be used as the method of training anomaly detection models for ICS. While performance of these models will offer significant improvements, they can also be a single point of failure if deployed on a single machine. In order to prevent these models from becoming single

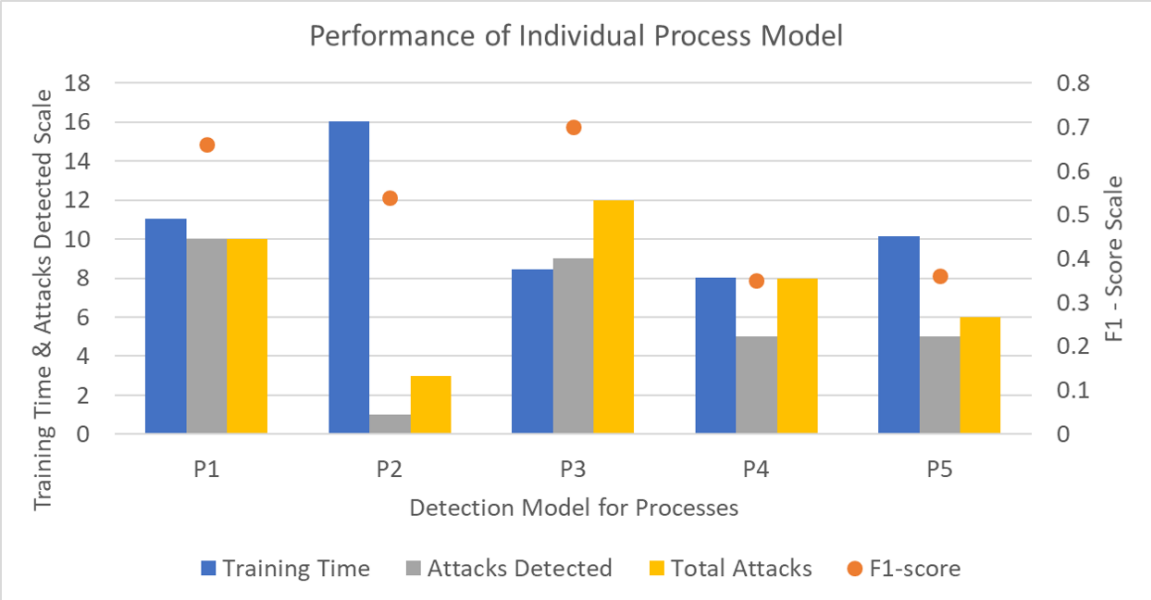


Figure 3.2: Performance of distributed detection models trained on individual processes.

points of failures, they should be implemented with multiple hot backups in different machines.

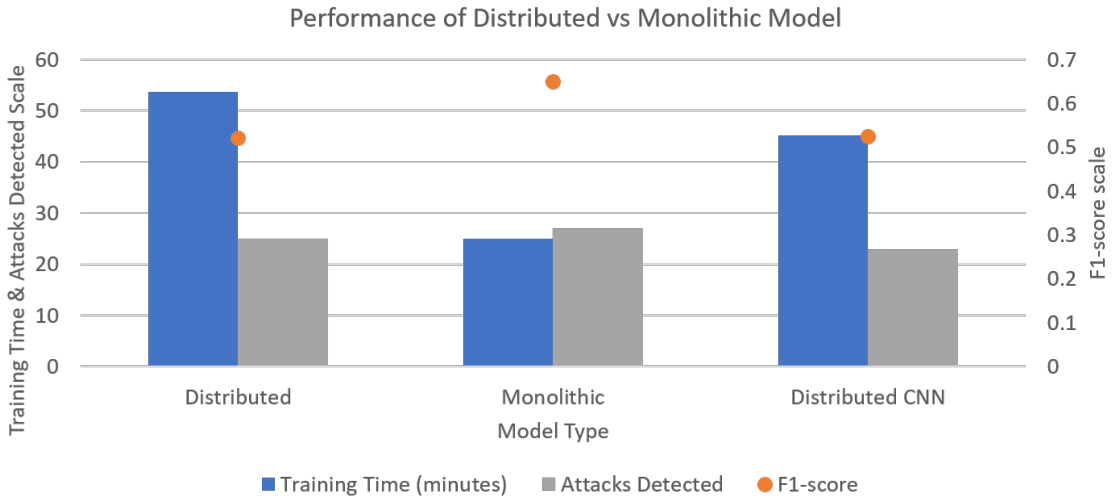


Figure 3.3: Comparative performance of distributed model and single individual model.

CHAPTER 4

RISK ANALYSIS

Anomaly detection mechanisms explored in this thesis provide an active monitoring security layer. Along with this monitoring, this thesis also proposes risk analysis through modelling physical processes, as well as attack scenarios for these systems. Here, risk analysis is the process of analyzing a system to identify potential threat events that can negatively impact a system and its assets. In the domain of ICS, this equates to conducting risk analysis to identify threats to an implemented system and the impact these threats can have to the processes. During design, development, and implementation of control systems, risk analysis helps engineers identify critical parts of the system, and provide information on criticality and threat severity of components. In order to facilitate this process, this chapter aims to provide a methodology of creating Colored Petri Net (CPN) models for the physical processes, and potential attack vectors to these processes. This methodology depends upon domain expert knowledge on the process control to build a physical process simulation utilizing CPN, as well as knowledge on identifying potential failure modes within the modelled simulation. By building these models, domain experts can conduct offline simulation of attacks and learn risks these attacks carry to the system.

4.1 Related Work

This section explains three forms of risk analysis proposed for network as well as ICS security: (I) Petri Nets, (II) Game Theory, and (III) Attack Trees.

Petri Nets are graphs where nodes represent transitions and places, directed arcs represent pre and post-conditions for transitions, and tokens in nodes represent the marking of current state within the places. Henry et al. proposed utilizing Petri Nets (PN) to carry out risk analysis in the SCADA implementation off ICS [24]. In their context, risk is defined as the function of resources an attacker can gain access to during an attack. To find these functions, PN's are built for the Remote Terminal Unit (RTU), and workstations. These PN's are then analyzed to form a set of devices and rules within them that must be available to an attacker for a successful attack. The focus of this body of work is in mapping risks on the communication protocols and rules within them to mark risks in SCADA architecture. Authors also form an assumption that each place in the petri net has a binary marking, which can not be true when real physical processes are being simulated. Along with these risks, there is need to study risks within the processes themselves.

Attack Trees (AT) are a conceptual tree structure that represent how assets could be attacked within a system. Each node in the tree represents a step an attacker must take to achieve their final goal. These tree structures allow for analysis of risks within a system, and find areas that can be protected to prevent an attacker from achieving their final goal. Byres et al. [8] propose building attack trees for the internal Supervisory Control and Data Acquisition (SCADA) system communication protocol and it's implementation to gain control of the administrative devices. A thorough analysis of the system is carried out by domain engineers to build attack trees of the

higher level attacker goals. These goals are then broken down into separate trees that conceptualizes each step an attacker might need to take. Multiple communication protocol risks were identified by studying the paths within the attack tree. Although this process was effective in identifying risks within the SCADA implementation, verification of coverage of attack trees to all components in a system is not possible. This can create incomplete attack trees and leave domain experts unaware of risks in the system.

In addition to attack trees, attack graphs can be used to analyse threats to a system. Unlike attack trees, nodes in an attack graph are a set of rules that must be followed for each communication taking place within the system. Ingols et al. [26] present a NetSPA attack graph system that is used to model threats and countermeasures in network communication. These models are created using network vulnerability scans and firewall rules. They are then used to create reachability and attack graphs representing paths attacker can take within the system by exploiting known vulnerabilities. Existence of extensive vulnerabilities database and network scanners within Information Technology (IT) domain helps NetSPA systems to be effective in building threat models for existing and hypothetical vulnerabilities. Extension of this methodology to ICS is hindered by the requirement of similar vulnerabilities databases for ICS, as well as better scan tools for the Operational Technologies (OT) in use.

Another approach to understanding risks within a system is through a game theoretic study of interactions between an attacker and administrator/Intrusion Detection System (IDS). Game theory allows us to build a strategic decision making process for administrator/IDS where the risks of attacker gaining control within the system are minimized. Through discovery of the nash equilibrium, it allows an administra-

tor/IDS to find an equilibrium strategy where the attacker and administrator/IDS both have the best gains. Knowing this equilibrium allows the system to be designed such that an attacker can not reach this equilibrium state and is always accruing losses. Applications of game theoretic models to netowrk security have been proposed by various authors [34, 5]. Lye et al. [34] propose a two-player stochastic game between an attacker and administrator with static information to find response strategies for administrator to protect their network system. On the other hand, Alpcan et al. [5] propose a two-player finite game with dynamic information, between attacker and an IDS that obtains information from multiple other IDS's. Both these papers aim to find response strategies, and security trade-offs of these strategies when a system is responding to an attacker. While this process is extremely effective at finding cost/gain relationships between actions taken, it requires a stochastic model of the actions and probability distribution for these actions to be defined by domain experts.

4.2 Methodology

This section presents the background of Petri Nets, and methodology used to create Petri Net models for ICS physical processes, and attack models.

4.2.1 Physical Process Model

Petri Net (PN) models are a directed graph with places and transitions connected through directed arcs. Formally, Petri Nets can be defined as a four-tuple $< P, T, I, O >$ where P represents the set of places [41]. Places are derived from the set of states for each component that needs to be modelled for a system. T represents a set of transitions, and are the possible events that change the state within the modelled

system. Input (I), and Output (O) map places to transitions. $I(t_i)$ is the input of a transition and $O(t_i)$ is the output obtained from firing these transitions. Each PN also contains markings, which is a definition of distribution of tokens within the different places within the net [24]. These markings help define the state of the places, and the transitions that can be fired. In a PN, transitions can only fire if the weight of the arc connecting the transition to the inputs matches the number of tokens in the inputs. When a transition fires, the tokens in the inputs are removed, and tokens are distributed to outputs based on the weights of the arcs connecting transitions to their outputs. The weights to the arcs can be defined based upon the requirements of the system. A sample PN representing a sequential process with three places and two transitions is shown in Figure 4.1. A simple PN allows simple unit tokens to be used as markings in the model. Modelling of complex systems requires usage of Colored Petri Nets (CPN) that allow usage of complex tokens such as sets, real numbers, etc. and allow usage of multiple types of tokens within the same place. In CPN, markings specify the collection of colored tokens where colors are user defined sets. This allows for complex message passing within a system to be modelled within a CPN, in turn helping model a complex physical process [19].

This thesis proposes utilizing CPN's to model the control logic followed by a Programmable Logic Controller (PLC) in ICS and building an attack model that relates to this CPN. CPN model of the control logic are used to simulate the physical process behavior of ICS. This is achieved by representing each state of actuators, and each sensor as places in the CPN. Sensors are modeled with a color set of $REAL$, and actuators are defined using a custom color set of ordered pair $(BOOLEAN, REAL)$. Transitions in CPN's are used to define the control commands and state changes happening in each sensor/actuator. Further, guards in transitions are used to define

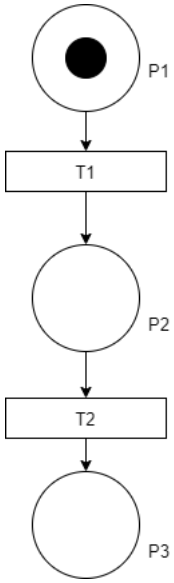


Figure 4.1: A sample Petri Net model of a sequential process.

IF statements used within PLC. These guards are important for transitions that need to be fired only when a place value reaches below a set point. Finally, arcs are used to model the information flow between components and set minimum weights at which transitions fire.

4.2.2 Attack Models

Once a CPN model is built for the physical process, it can be used to find and model attack vectors that deviate the system from normal state. In this thesis, attacks that change the physical states of the system by manipulating the sensors/actuators, and spoofing commands to the PLC are modelled. Each attack consists of a set of places that represent the type of process control manipulation modes such as manipulation of sensor, control code manipulation, etc. This process control manipulation modes of attacks are related to a set of physical manipulation modes. Here, physical manipulation modes are a set of places that are defined by physical changes in

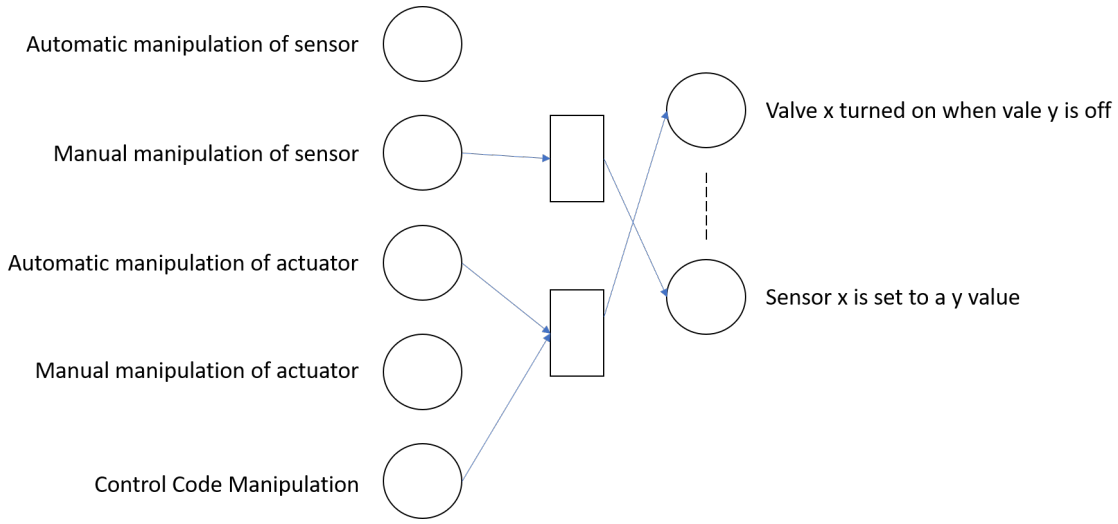


Figure 4.2: Petri Net model of control attack models coupled with physical manipulation modes

sensors/actuators that cause damage in the physical system. The process control manipulation modes and physical manipulation modes are related to each other by transitions. Pre-conditions of the transitions define the type of control failures an attacker needs to induce to successfully achieve the physical failure in the system. Figure 4.2 presents an example of building these attack models. As discussed, each place on left side of PN are the control manipulation modes, and on the right side of PN are physical manipulation modes. These manipulation modes can be identified based on the components used in each process and the PN must be built for each individual process. Physical manipulation modes are identified from the physical process model, and domain expert knowledge on state changes within the model that would cause damage to the system. Control manipulation modes will remain similar for most processes, as attacker can only cause manipulation modes such as automatic/manual manipulation of sensors/actuators, and change command or parameters sent to/form controller.

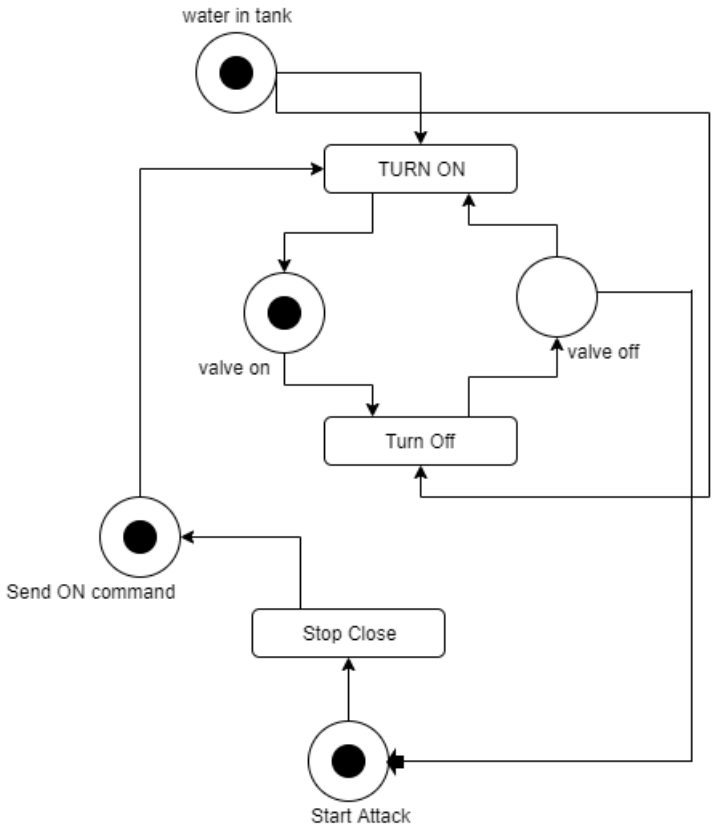


Figure 4.3: Example Petri Net attack model

Once a set of attack models are defined, these attacks are mapped to the physical process CPN. In this model, transitions represent the steps attackers took to start the attack process. Pre-conditions and post-conditions to the attacks are represented as places within the CPN. Here, pre-conditions help identify states of the system the attacker must be aware of to have successful impact. Once the attack transitions fire, they can pass through multiple places and transitions to signify the extra steps an attacker must take to have the desired impact within the system. For example, figure 4.3 presents an example attacker model for a hypothetical process. Here, a valve is turned ON or OFF depending upon the water level within a tank. The places for the hypothetical process this figure represents are the sensors (water in tank) and

states of the actuators (valve on, and valve off). Transitions are the actions taken based on the value within the sensors. Once a model is built, we can model attacks that can cause damage within the system. In a simple process of figure 4.3, one of the attacks can be spoofing actuator command to force a motor valve into turning on even if the water level is high on the tank. The transition "Stop Close" represents the start of attack, and place "Send ON command" represents the attacker action of sending command to the valve. Mapping this attack to the manipulation modes model built in figure 4.2, shows that an attacker needs control code manipulation or automatic manipulation of actuators to achieve their goal. Therefore, one risk to this hypothetical system is an improper state change in the valve achieved through control code manipulation or automatic actuator manipulation.

4.3 Implementation

This section presents the CPN models for SWaT tested, and the attack models built for it:

4.3.1 Physical Model of SWaT

Creating the Petri Net model for a physical process based on sensors/actuator behavior requires knowledge of the control logic. Therefore, this thesis only builds a CPN for P1 of SWaT testbed as it is the only process with the control logic publicly available. Analysis of the PLC code shows that only three components (LIT101, MV101, P101) are observed and affected by the controller. Therefore, the detailed CPN can be built for this process utilizing these three components and their states. As presented in Figure 4.4, the CPN model for P1 contains places representing the states of Pump

101 (P101), Motor Valve 101 (MV101), and Level Indicator (LIT101). Within this model, sensor LIT101 is defined as a place with REAL data type and the value of LIT01 can go from 0 to 1000. Four main transitions are defined to be fired based upon the markings within the LIT101 place. As defined in the control scheme of P1, P101 is turned ON when LIT is above 250mm, and MV101 is turned ON when LIT101 is under 250mm. These actuators are turned OFF when these conditions are not met. Hence, they are defined with a user defined color set of $\langle boolean,\ REAL \rangle$. Changes in markings are carried out by firing transitions, and arcs are used to represent the magnitudes of change as well as flow of data between various places and transitions. Weights in the CPN for P1 are defined based on the physical properties of the flow of water per second. Each transition counts as events happening within that second. Once defined, this model can be initialized with markings in individual places and simulated as necessary. This simulation is then verified to be correct by comparing against the data obtained from SWaT dataset.

4.3.2 Attack Models

Once the physical simulation for P1 is built, attack types CPN is created for P1 as described in section 4.2.2. As shown in figure 4.5, attacks are represented in terms of actions that need to be taken on the manipulations an attacker needs to excite in process control and the physical target of states of attacker. Within P1, control manipulation could be resulted by automated/manual manipulation of the sensors or manipulation of control code. Further, physical manipulations are defined on the right side of figure 4.5 based upon incorrect state changes occurring within the system. Each physical manipulation mode is connected to control manipulation mode by transitions. For example, manipulation of MV101 to turn ON while level

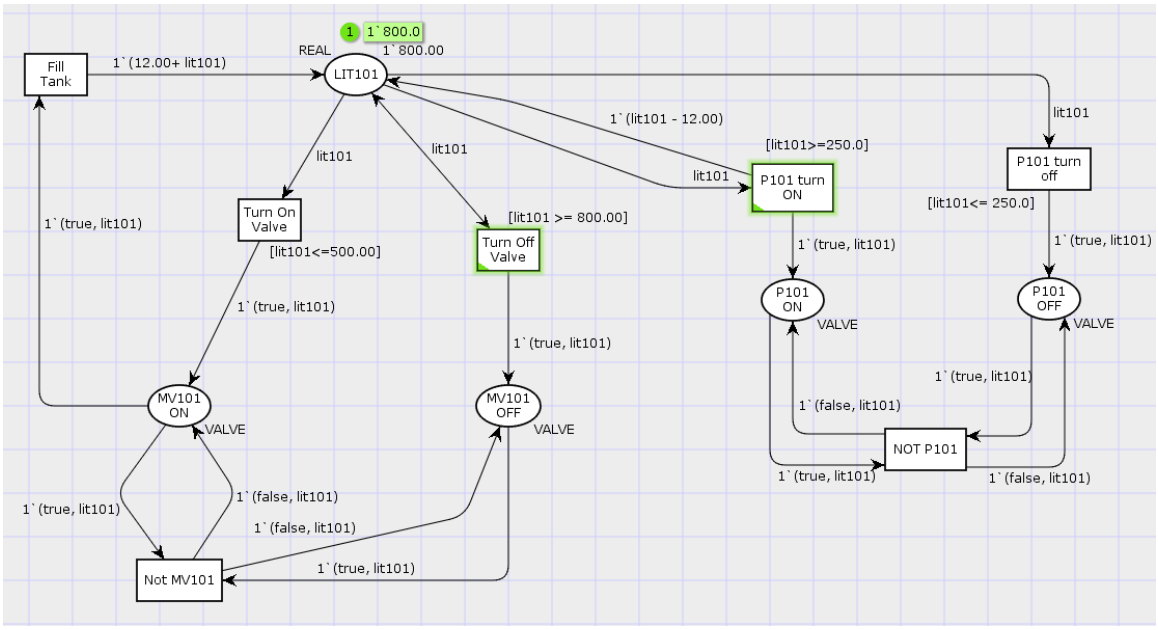


Figure 4.4: CPN model for Process 1 of SWaT testbed.

in tank is above the safe threshold is a manipulation obtained through automatic manipulation of actuators and control code manipulation. Here, an attacker does not need to manually instigate an attack as it can be launched automatically once LIT101 reaches a set point. Once the manipulation modes are identified, physical manipulations obtained from coupling of control manipulation modes are added to the physical process simulation CPN. Figure 4.6 presents the physical attack simulation created for two attacks to the system. While only two attacks are represented in the figure due to the space limitation in print, all the physical manipulation modes represented in figure 4.5 can be added to their respective points in the model. After creating a model with the respective physical manipulations in place, the CPN can be simulated to observe how the physical process reacts to the manipulation modes and evaluate the defined risks with real behavior of the physical processes. For example, during the manipulation mode of *ef1*, motor valve in the process turns ON while the

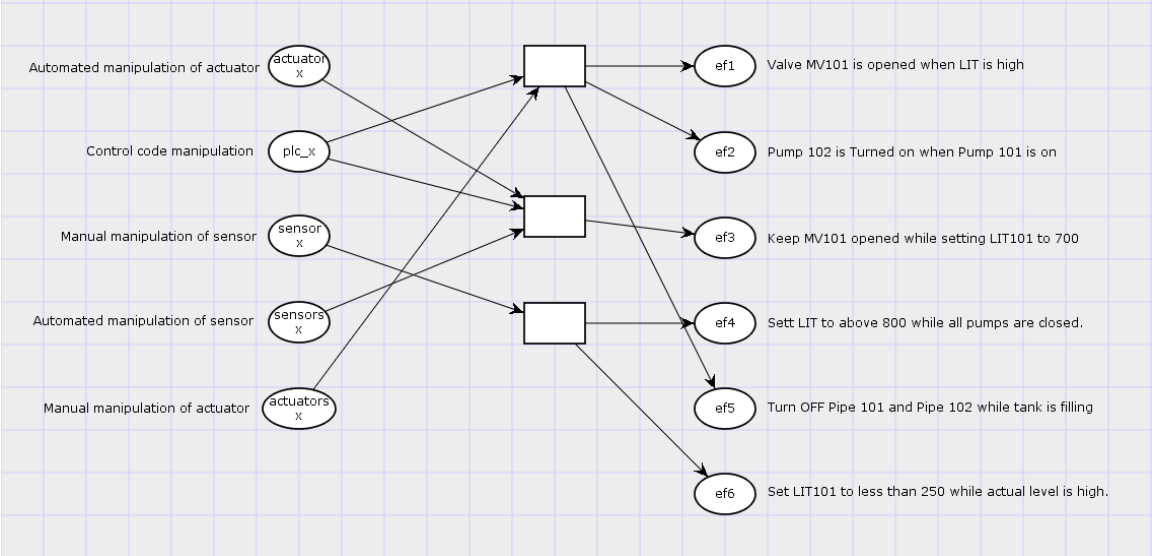


Figure 4.5: Attacks are represented in terms of their process control manipulation modes and physical manipulation modes.

LIT level is above defined threshold, but this fails to have an instant impact within the system as P101 is constantly pumping water out. So, this manipulation does not carry a high risk within the first few ticks, but after a period of time the level in LIT is shown to reach a dangerous level. Through simulation of physical process and manipulation modes modelled, this methodology can be used to identify these risks that can take an instant or a long period of time to have an impact within the system.

4.4 Conclusion and Future Work

Colored Petri Nets provide a mechanism for modelling physical processes such that they can be simulated for offline observation of a system. These models can be used to systematically identify potential attack points, and create attack models for an existing system. Creating these attack models based upon coupling of manipulation

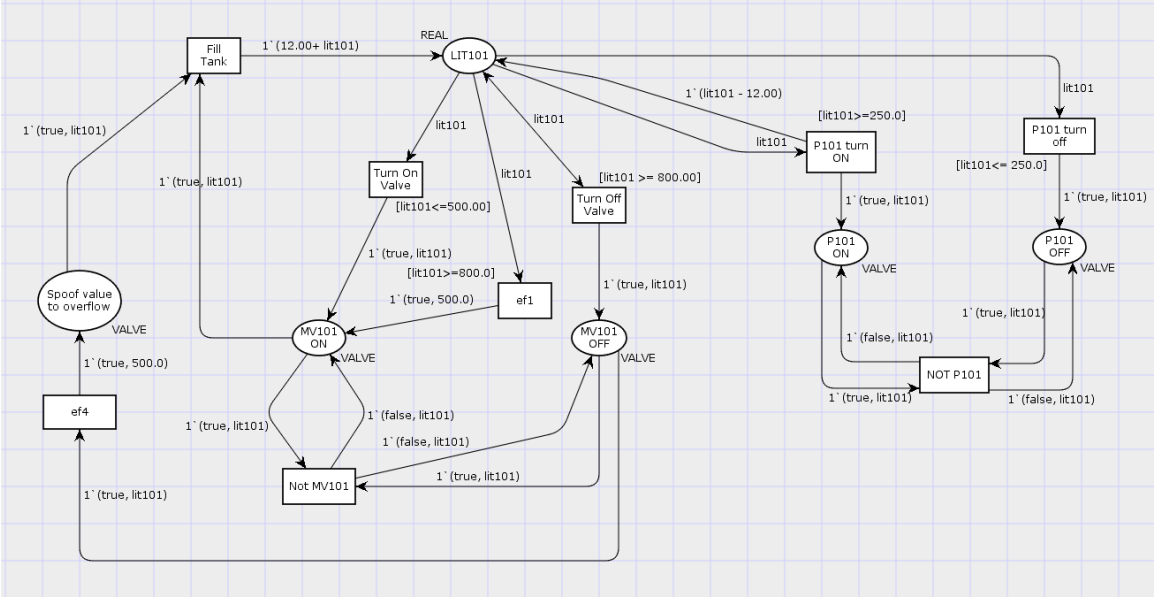


Figure 4.6: CPN showing two attacks to the physical process.

modes in controls and manipulation modes in physical states allows for domain experts to identify specific components to observe when searching for attacks. Furthermore, simulating identified manipulations within the created CPN models allows for identification of physical process behaviors under attack. Implementation of this methodology in process 1 of SWaT testbed showed correct simulation of the physical process. Further, simulation of P1 after the identification and addition of physical manipulation modes was successfully tested to analyze risks the manipulation modes carry in the physical process.

While it is not within the scope of this thesis, coverability analysis on Petri Net's can be used to identify sets of outcomes that are possible within the physical process for each unique initial marking on the net. Therefore, coverability analysis can help identify chain of transitions and markings that are possible, including attack markings based on the attack petri nets. To apply this analysis on the methodology discussed in this thesis, method of conducting coverability analysis for an unbounded CPN that

contain non-binary places need to be created. This body of work will be continued by the research team in the future.

CHAPTER 5

DISCUSSION

This thesis presents an anomaly detection model for ICS, the best architecture of training and deploying the detection models, and a CPN based simulation of processes under normal as well as attack conditions. The anomaly detection models in ICS have to be able to detect anomalies that are defined by outliers in data, as well as abnormal interactions of different components in the physical space. Unsupervised detection of these abnormal interactions in higher dimensional space has been shown to be inefficient by previous works in this domain. At the same time, semi-supervised methodologies require additional steps of statistical deviation detection where large number of hyper-parameters need to be defined. With the availability of fault data and labeling capabilities within industries, supervised methodologies can be leveraged to build anomaly detection models for these systems. Findings of chapter 2 provide evidence of supervised methodologies learning classification boundary between normal and anomalous behavior within ICS data despite the low amount of anomalous data available. In the relevant test-bed, supervised methodology detected 29 attacks while maintaining extremely low training as well as testing time. This has been achieved through the usage of Wavelet transformation to append information to existing data along with the introduction of LSTM and Convolution based methodology of training an anomaly detection system.

Along with building a methodology for training anomaly detection models in this domain, chapter 3 presents empirical results of training anomaly detection models on the entire process compared to distributed into individual processes. Individual process models showed varied effectiveness with some models performing better than others. As a whole, monolithic models are able to detect higher number of attacks with better F1-score due to their view of the global relationships between features in the data. This presents a difficulty in deployment of detection models in ICS as single models are a single point of failure. Deployment strategies for these trained models within the physical back-plane of controllers must be studied for a comprehensive analysis.

Finally, a Colored Petri Net based physical process and failure simulation models were introduced to assist in risk analysis. Here physical manipulations were coupled with control manipulations to model potential attack risks within a physical process. Simulations of these CPN's are then utilized to visualize the behavior of the process under these failure modes. These models are the first step in fine grained analysis of physical process control code and risks within an existing control system. While creation and implementation of CPN's for processes is a time intensive procedure, a working simulation built from existing PLC code can be a source of offline analysis and data collection mechanism.

While this thesis provides improvements in supervised deep learning architectures that can be used for creating anomaly detection models for ICS, it works under the assumption that attack labels can be added for the sensors/actuators data. There are various ways of obtaining this data, including labelling faults in the data-set as attacks. At the process level, faults are going to have the same characteristics as an attack as they are violating the normal procedure of the process. Labelled data can

also be collected by simulating attacks with methodologies such as Honeypots[45], simulations[13, 21], and deep learning attack generation techniques [16]. While these are simulations of working ICS, they offer data that can be used to build an extra layer of protection for these systems. These assumptions should be tested to verify the complete life-cycle of building supervised deep learning detection methodologies for ICS.

CHAPTER 6

CONCLUSIONS

This thesis explored a LSTM + CNN based supervised anomaly detection methodology for ICS that catches 27 out of 36 attacks in SWaT testbed while maintaining a low cost training time. It introduced a wavelet transformation feature engineering pipeline for data pre-processing to encode information for training of anomaly detection models in this domain. Furthermore, various deep learning as well as unsupervised methodologies were explored for a comprehensive comparison. Through this work, we observe higher performance for supervised methodology compared to previous unsupervised as well as semi-supervised deep learning methodologies applied to the data-set. In addition, this thesis explored the effectiveness of training a monolithic model and distributed models within ICS. This study concluded that distributed models suffer in performance and require large training resources when compared to monolithic models. Further, CPN based physical process and manipulation modelling techniques were used to create risk analysis and simulation models.

The findings of this thesis paves way for future research in the field of anomaly detection within ICS. Results obtained within the test data-set can be tested against other data-sets to verify the extensible nature of the anomaly detection approach. Similarly, approaches resulting in identification of attacked components should be explored to provide systems engineers with context within the detect time-period.

CPN simulation methods described can be extended to include coverability set generation to automatically mark potential risks within a CPN built by domain experts. Further, an ontology and Description Logic based model checking anomaly detection methodology can be built to provide context to attacks detected.

REFERENCES

[1] Marshall Abrams and Joe Weiss. Malicious control system cyber security attack case study–maroochy water services, australia. *McLean, VA: The MITRE Corporation*, 2008.

[2] Sridhar Adepu and Aditya Mathur. Assessing the effectiveness of attack detection at a hackfest on industrial control systems.

[3] Sridhar Adepu and Aditya Mathur. Distributed detection of single-stage multipoint cyber attacks in a water treatment plant. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 449–460. ACM, 2016.

[4] Chuadhry Mujeeb Ahmed and Aditya P Mathur. Hardware identification via sensor fingerprinting in a cyber physical system. In *Software Quality, Reliability and Security Companion (QRS-C), 2017 IEEE International Conference on*, pages 517–524. IEEE, 2017.

[5] Tansu Alpcan and Tamer Basar. A game theoretic approach to decision and analysis in network intrusion detection. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, volume 3, pages 2595–2600. IEEE, 2003.

[6] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271, 2018.

[7] Rafael Ramos Regis Barbosa and Aiko Pras. Intrusion detection in scada networks. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 163–166. Springer, 2010.

[8] Eric J Byres, Matthew Franz, and Darrin Miller. The use of attack trees in assessing vulnerabilities in scada systems. In *Proceedings of the international infrastructure survivability workshop*, pages 3–10. Citeseer, 2004.

[9] Defense Use Case. Analysis of the cyber attack on the ukrainian power grid. *Electricity Information Sharing and Analysis Center (E-ISAC)*, 2016.

[10] Marco Caselli, Emmanuele Zambon, and Frank Kargl. Sequence-aware intrusion detection in industrial control systems. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, pages 13–24. ACM, 2015.

[11] Yuqi Chen, Christopher M Poskitt, and Jun Sun. Learning from mutants: Using code mutation to learn and monitor invariants of a cyber-physical system. *arXiv preprint arXiv:1801.00903*, 2018.

[12] Ingrid Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE transactions on information theory*, 36(5):961–1005, 1990.

[13] CM Davis, JE Tate, H Okhravi, C Grier, TJ Overbye, and D Nicol. Scada cyber security testbed development. In *2006 38th North American Power Symposium*, pages 483–488. IEEE, 2006.

[14] James P Farwell and Rafal Rohozinski. Stuxnet and the future of cyber war. *Survival*, 53(1):23–40, 2011.

[15] Davide Fauri, Daniel Ricardo dos Santos, Elisa Costante, Jerry den Hartog, Sandro Etalle, and Stefano Tonetta. From system specification to anomaly detection (and back). In *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 13–24. ACM, 2017.

[16] Cheng Feng, Tingting Li, Zhanxing Zhu, and Deeph Chana. A deep learning-based framework for conducting stealthy attacks in industrial control systems. *arXiv preprint arXiv:1709.06397*, 2017.

[17] Igor Nai Fovino, Andrea Carcano, Thibault De Lacheze Murel, Alberto Trombetta, and Marcelo Masera. Modbus/dnp3 state-based intrusion detection system. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 729–736. IEEE, 2010.

[18] James B Fraley and James Cannady. The promise of machine learning in cybersecurity. In *SoutheastCon 2017*, pages 1–6. IEEE, 2017.

[19] Vijay Gehlot and Carmen Nigro. An introduction to systems modeling and simulation with colored petri nets. In *Proceedings of the Winter Simulation Conference*, pages 104–118. Winter Simulation Conference, 2010.

[20] Felix A. Gers, Jrgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471, 1999.

[21] Annarita Giani, Gabor Karsai, Tanya Roosta, Aakash Shah, Bruno Sinopoli, and Jon Wiley. A testbed for secure and robust scada systems. *SIGBED Review*, 5(2):4, 2008.

[22] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. A dataset to support research in the design of secure water treatment systems. In *International Conference on Critical Information Infrastructures Security*, pages 88–99. Springer, 2016.

[23] Jonathan Goh, Sridhar Adepu, Marcus Tan, and Zi Shan Lee. Anomaly detection in cyber physical systems using recurrent neural networks. In *High Assurance Systems Engineering (HASE), 2017 IEEE 18th International Symposium on*, pages 140–145. IEEE, 2017.

[24] Matthew H Henry, Ryan M Layer, Kevin Z Snow, and David R Zaret. Evaluating the risk of cyber attacks on scada systems via petri net analysis with application to hazardous liquid loading operations. In *2009 IEEE Conference on Technologies for Homeland Security*, pages 607–614. IEEE, 2009.

[25] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[26] Kyle Ingols, Matthew Chu, Richard Lippmann, Seth Webster, and Stephen Boyer. Modeling modern network attacks and countermeasures using attack graphs. In *2009 Annual Computer Security Applications Conference*, pages 117–126. IEEE, 2009.

[27] Jun Inoue, Yoriyuki Yamagata, Yuqi Chen, Christopher M Poskitt, and Jun Sun. Anomaly detection for a water treatment system using unsupervised machine learning. In *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*, pages 1058–1065. IEEE, 2017.

[28] Khurum Nazir Junejo and Jonathan Goh. Behaviour-based attack detection and classification in cyber physical systems using machine learning. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, pages 34–43. ACM, 2016.

[29] Eunsuk Kang, Sridhar Adepu, Daniel Jackson, and Aditya P Mathur. Model-based security analysis of a water treatment system. In *Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems*, pages 22–28. ACM, 2016.

[30] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. Multivariate lstm-fcns for time series classification. *Neural Networks*, 116:237–245, 2019.

[31] Sarfraz Khurshid, Darko Marinov, and Daniel Jackson. An analyzable annotation language. In *ACM SIGPLAN Notices*, volume 37, pages 231–245. ACM, 2002.

[32] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[33] Qin Lin, Sridha Adepu, Sicco Verwer, and Aditya 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*, pages 525–536. ACM, 2018.

[34] Kong-wei Lye and Jeannette M Wing. Game strategies in network security. *International Journal of Information Security*, 4(1-2):71–86, 2005.

[35] Aditya P Mathur and Nils Ole 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)*, pages 31–36. IEEE, 2016.

[36] ICS-CERT MONITOR. Ics-cert monitor. `https://ics-cert.us-cert.gov/sites/default/files/Monitors/ICS-CERT_Monitor_Nov-Dec2016_S508C.pdf`, 2017.

[37] Thomas Morris, Rayford Vaughn, and Yoginder Dandass. A retrofit network intrusion detection system for modbus rtu and ascii industrial control systems. In *2012 45th Hawaii International Conference on System Sciences*, pages 2338–2345. IEEE, 2012.

[38] John Mulder, Moses Schwartz, Michael Berg, Jonathan Roger Van Houten, Jorge Mario, Michael Aaron King Urrea, Abraham Anthony Clements, and Joshua Jacob. Weaselboard: zero-day exploit detection for programmable logic controllers. *Sandia report SAND2013-8274, Sandia national laboratories Google Scholar*, 2013.

[39] A Kaung Myat. Secure water treatment testbed (swat): An overview, 2015, 2016.

[40] Executive Order. 13636. *Improving Critical Infrastructure Cybersecurity*, 2(7), 2013.

[41] James L Peterson. Petri nets. *ACM Computing Surveys (CSUR)*, 9(3):223–252, 1977.

[42] Dmitry Shalyga, Pavel Filonov, and Andrey Lavrentyev. Anomaly detection for water treatment system based on neural network with automatic architecture optimization. *arXiv preprint arXiv:1807.07282*, 2018.

[43] Muhammad Azmi Umer, Aditya Mathur, Khurum Nazir Junejo, and Sridhar Adepu. Integrating design and data centric approaches to generate invariants for distributed attack detection. In *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 131–136. ACM, 2017.

[44] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, page 125, 2016.

[45] Emmanouil Vasilomanolakis, Shreyas Srinivasa, Carlos Garcia Cordero, and Max Mühlhäuser. Multi-stage attack detection and signature generation with ics honeypots. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pages 1227–1232. IEEE, 2016.

[46] Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yanmiao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. Machine learning and deep learning methods for cybersecurity. *IEEE Access*, 6:35365–35381, 2018.