Fakultät für Informatik
Technische Universität München

Anomaly Detection for Cyber-Physical Systems

Peter Schneider

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:

Prof. Dr.-Ing. Jörg Ott

Prüfende der Dissertation:

1. Prof. Dr. Claudia Eckert
2. Prof. Dr.-Ing. Georg Sigl

Die Dissertation wurde am 19.04.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 06.08.2021 angenommen.

Abstract

Tighter integration of technology in our everyday life is an omnipresent phenomenon. Consequentially, operators of industrial systems also adopt the possibilities of new technology. To enable many of the emerging use case, it is often necessary to adapt the usage of involved cyber-physical systems (CPS) far beyond their original design. Thereby, the ever-increasing number of connections between these systems and to the Internet make them vulnerable because of an increased attack surface. Due to the coupling with the physical world, detecting security incidents especially in CPS too late can lead to severe damages. Here, anomaly and intrusion detection to discover these incidents is an important prerequisite enabling early incident response. Hence, we show how to transfer and adapt existing detection systems into the industrial domain.

Though, the lacking data basis in industrial applications often inhibits the use of existing detection systems as these are usually required. Therefore, we investigate which data is useful and provide two CPS-specific methods to acquire needed information for current intrusion detection solutions. To ease the direct capturing of real-world data, we suggest a lightweight and efficient compression mechanism dedicated to industrial network data. As direct capturing is not possible in all use cases, we demonstrate the derivation of suitable datasets from simulations of the underlying processes.

For the actual anomaly detection, we provide a new method able to cope better with the environment found in CPSs. Using deep learning, we construct a method for high-performance feature learning and anomaly detection suitable for various industrial fieldbus protocols. Although not requiring any information on the encoding of data in the protocols, we can achieve detection rates for specific attack types of up to 99%.

Building on these basic blocks, we analyze the behavior of distributed detection solutions in CPS. Our method based on simulations of complex CPS networks allows for evaluating specific setups as well as for optimizing them. By this, we are able to derive optimized setups for shortest times until detection or highest detection rates under ressource constraints.

Overall, this yields fundamental insights in the application of anomaly and intrusion detection system in the industrial domain. Applying these in real-world setups allows

for an earlier detection by operators and can, therefore, limit possible high damages due to security incidents by early incident response.

Zusammenfassung

Die engere Integration von Technologie in unser Alltagsleben ist ein allgegenwärtiges Phänomen. Entsprechend wenden auch Betreiber industrieller Systeme die Möglichkeiten dieser neuen Technologien an. Um viele der dabei enstehenden Anwendungsszenarien umzusetzen, ist es oft notwendig die Verwendung involvierter cyber-physischer Systeme (CPS) weit über die usprüngliche Designidee der Systeme hinaus anzupassen. Hierbei macht sie die ansteigende Anzahl von Verbindungen zwischen den Systemen und zum Internet durch eine immer größere Angriffsfläche verwundbar. Zu spät erkannte Sicherheitsvorfälle können dabei insbesondere bei CPS durch die Kopplung mit der realen Welt oft zu hohen Schäden führen. Dabei ist Anomalie- und Angriffserkennung zur Entdeckung von diesen Vorfällen eine wichtige Voraussetzung, um frühzeitiges Handeln zu ermöglichen. Daher stellen wir da, wie existierende Erkennungssysteme in das industrielle Umfeld übertragen und angepasst werden können.

Eine im industriellen Umfeld unzureichende Datenbasis verhindert jedoch den Einsatz bestehender Erkennungssysteme, die eine solche in der Regel voraussetzen. Wir untersuchen daher, welche Daten nützlich sind, und stellen zwei für CPS spezifische Methoden vor, um benötigte Informationen für aktuelle Erkennungssysteme zu sammeln. Um die Aufzeichnung von realen Daten zu vereinfachen, schlagen wir ein leichtgewichtiges und effizientes Kompressionsverfahren speziell für industrielle Netzwerkdaten vor. Da die Aufzeichnung nicht in allen Anwendungen möglich ist, zeigen wir die Ableitung von passenden Datensätzen aus Simulationen der zugrundeliegenden Prozesse.

Zur eigentlichen Anomalieerkennung liefern wir eine neue Methode, die auf die Umgebung von CPS spezialisiert ist. Durch die Nutzung von Deep Learning konstruieren wir ein Verfahren zu hochperfomantem Feature-Lernen und Anomalieerkennung angepasst für diverse industrielle Feldbusprotokolle. Obwohl das Verfahren ohne Informationen über die Kodierung von Daten innerhalb der Protokolle auskommt, können wir für bestimmte Angriffstypen Erkennungsraten von bis zu 99% erreichen.

Aufbauend auf diesen elementaren Teilen analysieren wir das Verhalten von verteilten Detektionslösungen für CPS. Unsere Methode basierend auf Simulationen komplexer CPS Netzwerke erlaubt die Auswertung und Optimierung spezifischer Konfiguratio-

nen. Dadurch können wir auf die Zeit zur Erkennung oder eine hohe Erkennungsrate optimierte Konfigurationen unter Ressourceneinschränkungen ermitteln.

Insgesamt liefert dies grundlegende Einsichten in die Anwendung von Anomalie- und Angriffserkennung im industriellen Umfeld. Die Anwendung dieser in realen Systemen ermöglicht den Betreibern eine frühere Erkennung von Angriffen und Störfällen und kann daher den möglicherweise hohen Schaden infolge eines Sicherheitsvorfalls durch frühzeitiges Handeln beschränken.

Acknowledgements

This thesis is the result of several years of effort. As with all projects in life there can only little be achieved if one person is supposed to do it alone. Many people supported me one way or the other to finally arrive at this stage of the document.

First of all, my wife and my sweet daughter really pushed me not to quit but to get this finished. Many thanks to you both, you have been incredibly worthy listeners.

Also, a big thanks to my parents and especially my grandparents who always believed in me and me getting the work done. Your support was tremendous and your histories guided me the way through all this. Ein großes Dankeschön auch an meine Eltern und besonders meine Großeltern, die immer an mich und die Vollendung dieser Arbeit geglaubt haben. Eure Unterstützung war großartig und eure Geschichten haben mir den Weg gezeigt.

Thanks to Claudia Eckert, this work is now more than just a random compilation of ideas but a coherent work on the topic. To all my former colleagues at the Fraunhofer AISEC: Thank you so much! Some of you have given advice from your own experience others just helped with a short coffee break. You probably cannot imagine how important this is for such a work. I want to thank especially Dieter who literally introduced me to scientific working and more or less explained me how research projects work. Dear PIN and CST members, you are such a great crew. I really appreciate your knowledge and input to questions. Alex, Sven, Stefan, Konstantin, Gerhard, Norbert, Ferdinand, Jan-Philipp, Hannah, Bartol, thank you a lot for all the support and enabling opportunities for my research. You all made quite a contribution to enabling this thesis. I am really glad to have such competent colleagues and good friends after all that time.

I also want to thank my mentor, Alexander Wolff, for the nice discussions and giving me an opportunity explaining my work to people outside the field.

Finally, also a big thank you to Carsten Schürmann and Peter Sestoft who supported my thesis without having any obligation to do so.

Contents

List of Figures

List of Tables

Acronyms

Blob	Binary Large Object.
CIP	Common Industrial Protocol.
COD	CANopen Object Dictionary.
COTS	commercial-off-the-shelf.
CPS	Cyber-physical System.
CTF	Capture-the-Flag.
CVE	Common Vulnerabilities and Exposures.
DMZ	Demilitarized Zone.
DNN	Deep Neural Network.
GPU	Graphical Processing Unit.
HDFS	Hadoop Distributed File System.
HIDS	Host Intrusion Detection System.
ICS	Industrial Control System.
IDS	Intrusion Detection System.
IIoT	Industrial Internet of Things.
IoT	Internet of Things.
IPS	Intrusion Prevention System.
JSON	JavaScript Object Notation.
NIDS	Network Intrusion Detection System.
NLP	Natural Language Processing.
OT	Operational Technology.

PCN	Plant Control Network.
PDO	Process Data Object.
PKI	Public Key Infrastructure.
PLC	Programmable Logic Controller.
RAM	Random Access Memory.
SCADA	Supervisory Control and Data Acquisition.
SDA	Stacked Denoising Autoencoder.
SIEM	Security Information and Event Management.
SRA	Security and Risk Assessment.
TPM	Trusted Platform Module.

1 Introduction

In the beginning of this thesis, we describe the motivation and major challenges tackled. From this, we then derive our research questions and list the contributions.

1.1 Motivation

What are CPS?

The emerging trend to interconnect field devices and control hardware through to office IT infrastructure, sometimes referred to as the fourth industrial revolution, leads to the creation of complex Cyber-physical Systems (CPSs) [1]. These systems steer real-word physical processes based on sensor measurements and input data. As the basic principle of cyber-physical systems is the orchestrated operation of software, hardware, actuators, and sensors, they combine the characteristics of algorithms and data with the physics of the controlled processes.

Security in CPS

So far, CPSs were used in dedicated and isolated environments. With the development of the fourth industrial revolution and the field of Industrial Internet of Things (IIoT), operators remove these air-gaps resulting in complex systems connected not only in local networks but even worldwide across country and company borders. Thus, these systems now experience the same risks as all other internet-connected devices do [2]. However, often lacking suitable security controls, they are still not prepared for the internet threat landscape.

Poor upgradability

Often, companies cannot fix existing security problems by updates. While security updates are already problematic in business IT, long life cycles and expensive safety certifications inhibit fast-paced software updates even more for CPSs. In addition, CPS life-cycles of up to decades further increase the problem in the absence of updates. Hence, we find these systems with outdated security posture accessible on the Internet.

Recent attacks on CPS

Unfortunately, even implementing currently available security controls in CPSs could not prevent all attacks. In the past, several major attacks have been reported, such as the Stuxnet attack [3] on an Iranian nuclear reactor, the uncontrolled shutdown of a blast furnace in Germany, and the power grid attack in Ukraine [4]. As these applications represent part of the backbone of our society, already small disturbances

in their operation may result in severe impacts for large groups of people. These attacks were tailored to specifically target systems with high damage potentials. An analysis of the Stuxnet attack [5] revealed that it even used the manipulation of application-specific project files to spread throughout the system. A modification of used programming libraries then finally led to the manipulation of the physical process.

Such attacks, also referred to as programmable logic controller (PLC) exploitation, manipulate the behavior of part of the CPS so that reactions of its communication partners are steered into the desired direction. While CPSs definitely also face the same threats as business IT, the high possible damage resulting from targeted attacks makes them more important to be act upon.

Securing CPS

Recent research on security measures for CPS focuses on transferring the lessons-learned from classical business IT to the industrial domain. They include the adaptation of CPS-specific protocols, filtering traffic with firewalls, as well as deploying intrusion detection systems [6]. We already showed that due to problematic update procedures and long life-cycles we must still expect attacks even when current security measures are implemented. Therefore, it is very important to at least limit the impact of these attacks by enabling an early incident response. Hence, there is a need for accurate and fast detection of security and operation anomalies for CPS. An ideal system is able to provide details to the severity of an incident, to its reason and to the question whether a safe operation is still possible — not just for a single controller but for a complete distributed setup of a CPS. However, current signature-based intrusion detection systems are limited in their effectiveness. Thus, developing systems striving to distinguish anomalies from the normal behavior of a system became a fundamental task [7].

Differences in IDS for CPS

Current anomaly detection mainly focuses either on the analysis of known attacks or on the analysis of metadata. In industrial settings, these approaches are either inapplicable or only use a subset of available information. The metadata in these systems is often static, can easily be analyzed, and correspondingly manipulated. On the other hand, neglecting the coupling with the physical worlds in CPS ignores half of the system's process. Usually, the parameters of the physical process will only be available in the application logic or the payload of network packets. With most commercial-off-the-shelf (COTS) Intrusion Detection System (IDS) focusing on metadata and network packet headers, this information is often completely ignored in detection systems. This way, cyber-physical systems differ from daily business IT, which does not directly interact with the physical reality. Therefore, they also require new methods to secure

their operation. While yet unknown attacks are linked to severe risks, the analysis of only known attacks does not provide methods for detecting new variants.

One approach to ameliorate the detection of such previously unknown and newly emerging attacks are anomaly detection systems based on multiple data sources or learning algorithms.

State-of-the-art

Although, researchers continually propose new detection algorithms, they are so far not optimized for operation in cyber-physical systems. Typically, low-power sensors and actuators used in CPS lack the resources to execute current anomaly detection methods. Further, the added characteristics of the physical world are usually not considered although they resemble part of the system. This situation suggests using multiple heterogeneous data sources to improve the detection quality. While in traditional office IT the processes are often independent of external influences, a cyber-physical system may behave differently given external parameters as temperature, humidity, time, or power. There are four main challenges to ameliorate that situation:

Challenges

1. Datasets

First, there are so far only a few comprehensive datasets to develop and test anomaly detection systems on cyber-physical systems. However, these are strong requirements to evaluate new ideas and to compare the performance and effectiveness of different approaches.

2. Framework

Second, we need a comprehensive framework for the whole anomaly detection process. As common business IT frameworks do not interact with the physical world, they usually lack providing corresponding characteristics and sensor data to the detection system.

3. Detection Method

Third, the detection algorithms themselves need to be optimized for their specific application. As cyber-physical systems are typically unique, an approach to building such adapted detection systems is needed to be applicable for real-world scenarios.

4. Staged Attacks

Fourth, anomaly detection systems based on one single data acquisition point give only local insights. As described before, recent attacks on CPS use multiple different attack stages and chain them for the final exploit. However, the most likely causes of single anomalies in cyber-physical systems, e. g. degradation of hardware, bad parameters, and physical uncertainties, are not relevant or even existent in business IT. For a proper distinction of actual intrusions among all process anomalies, the cause estimation should be rethought. A reliable method for identifying cause–effect relationships in CPS also yields additional insights for other problems like performance optimization, predictive maintenance, and big data analytics. Therefore in the end, such a method enhances our understanding of ongoing and future attack waves against current CPSs.

1.2 Research Questions

Nowadays, anomaly detection systems in industrial settings suffer from missing input data. As these systems build models out of provided input data, this data directly influences their detection rates. Hence, effective research is only possible if suitable data is available. Thus, many aspects like developing advanced detection systems and analysis of ongoing attack strategies have not been much explored so far. Therefore, the four previously stated challenges lead to the following major research questions:

- What is required to build an applicable intrusion detection framework for industrial settings?
- How can suitable training and testing data for anomaly detection systems in industrial settings be gathered?
- How can we use characteristic properties of industrial processes to enhance anomaly detection?
- How can we understand the course of ongoing attack strategies?

Building blocks

In order to find answers to these questions, several building blocks are needed. While each of them solves a different portion of the problem, a later integration into a complete framework yields a comprehensive anomaly detection solution appropriate for industrial settings.

Data generation

As it is challenging to acquire data from real-world cyber-physical systems, appropriate simulations can generate data needed for this thesis. Additionally, this allows for developing anomaly detection systems for facilities not yet built but just being planned. Furthermore, a simulation allows for deriving data from attack scenarios without the need to put expensive setups at risk.

Detection

To answer the third and fourth question, a suitable method for anomaly detection needs to be found which, on the one hand, can operate on the available data, and, on the other hand, enables insights about the causes and the development of attacks.

1.3 Contributions

This thesis strives to understand the needs of anomaly detection in cyber-physical systems. Every IDS is developed around three integral parts: the data analyzed, the method used, and the architecture or framework used for processing. These three items, however, are only building blocks for efficient anomaly detection in industrial use cases. Thus, we need to link them and understand their interplay as shown in Figure 1.1.

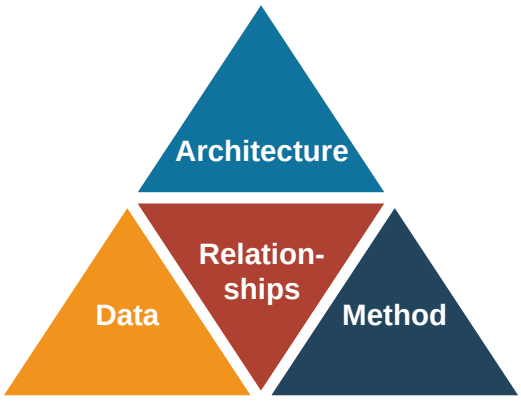

Figure 1.1: A high-level view of the contributions in this thesis.

As outlined in the previous section, every one of these cornerstones lacks suitable solutions for industrial systems. Throughout researching the individual building blocks, seven peer-reviewed publications [8, 9, 10, 11, 12, 13, 14] emerged.

Contribution 1 (**C1**) sums up how to transfer anomaly detection architectures into industrial use cases. **C2**, then tackles the problem of missing experimental data for advancing the field of intrusion detection in cyber-physical systems while **C3** adds a new high-performance method for anomaly detection.

Finally, we link all these cornerstones by analyzing the relationships between data, methods, and architecture. Thereby, in **C4**, we derive recommendations for the next generation of distributed anomaly detection systems.

C1: Architecture for Anomaly Detection in CPS Until nowadays, deriving correct and meaningful anomaly detection rules is a tedious manual task. While several architectures for detection of specific threats exist, there is no comprehensive guidance on which to use when and how. Therefore, we describe a method for deriving monitoring measures from the outcome of a security and risk assessment. Using a catalog of detection and monitoring techniques, we then derive needed data sources, preprocessing techniques and propose an architecture for aggregating this data. This process leads to a framework of merging different monitoring techniques and, thus, addresses the second posed challenge.

C2: Circumventing the Missing Data Problem One major problem for the development of new anomaly detection systems in industrial use cases is the lack of suitable training and testing data. While nearly all currently existing approaches rely on training data, only a limited set of publicly accessible data is available.

Towards solving this problem of missing industrial datasets, i. e. the first challenge, we provide two new methods for testing new industrial intrusion detection systems and introduce few already existing datasets.

2.1 We describe an architecture for directly capturing industrial data using Fieldbus testbeds. This architecture also allows the integration of real hardware in the loop as well as virtualized devices. We then extend this architecture to large scale applications using advanced compression techniques. We evaluate the de-duplication-based compression method on several industrial network traffic datasets. With this method, we achieve a reduction of data to be transferred of up to 64%.

2.2 We present a workflow and framework for simulating industrial network traffic. Therein, we exemplary show how to integrate attacks into the simulation model. We evaluate and investigate the usability of the generated data with state-of-the-art anomaly detection methods.

C3: High-Performance Anomaly Detection in CPS Current network anomaly detection methods focus on metadata analysis. Recent attacks, however, succeeded by stealthily manipulating the network packets' payloads. Hence, the actual manipulation is undetectable for them. On the other hand, current classical IT anomaly detection methods considering also packet payloads lack the needed performance for the ever-increasing network traffic in cyber-physical systems. Addressing the third challenge, we introduce a high-performance processing framework and method for industrial network anomaly detection. The framework is based on automated feature-learning for network packets independent of components and topology using stacked denoising auto-encoders. Our evaluation on two datasets using different Fieldbus network protocols yields $f1$-scores of over 99%. Additionally, we provide an approach for semi-automated labeling of unlabeled network traffic datasets.

C4: Understanding Cause–Effect Relationships Critic on the usage of machine-learning based anomaly detection systems rises from their often unexplainable results. For more understandable results and addressing the fourth challenge, we show how to aggregate information from several alert log types and formats throughout a distributed anomaly detection setup. Given an architecture for aggregating this information and a simulation framework, we can better understand the interplay of the different system parameters. An analysis of different parameter configurations allows for deriving best-practices and recommendations for future

installations of distributed anomaly detection systems. Thereby, we derive design rules for future distributed anomaly detection systems. Based on this information, we present methods for optimizing the placement of intrusion detection systems throughout a network. Thereby, we consider the specific situation by integrating the IDS' performance, the assumed attacker model, the local network architecture, and the resulting costs. This enables a reproducible way to estimate and handle the trade-offs when configuring IDS.

1.4 Structure

This thesis is structured as follows:

Background In Chapter 2, we provide definitions and background knowledge on relevant topics such as anomaly detection and cyber-physical systems. Relevant related work to the topics addressed in this thesis and how we expand on it is summarized in Section 2.4. Additionally, we differentiate anomalies from intrusions and give short explanations of machine-learning techniques used in this thesis.

The following chapters outline the process of anomaly detection for cyber-physical systems and deepen specific parts of it.

Anomaly Detection Process First, we give a high-level overview of a process for anomaly detection in cyber-physical systems in Chapter 3. Based on a security and risk assessment, we derive suitable monitoring measures for industrial systems. For each combination of a security objective (confidentiality, integrity, availability) and an asset (function, connection, data, component) we match suitable monitoring measures. Furthermore, we discuss how each of these measures can be implemented. We start by indexing possible data sources, give hints on possibly needed transformations of this data, and describe which source suits which measure.

The following chapters deepen the single steps of the previously described process.

Data Acquisition In Chapter 4, we present two new approaches for data acquisition to train and execute anomaly detection systems in cyber-physical systems.
First, we present an approach to scale this technique to larger networks incorporating advanced network traffic compression techniques in Section 4.1. Here, we describe an architecture for direct data capturing as well as a method for high-performance data transport of industrial network captures.

Second, we describe a method based on co-simulation for generating necessary training and testing data in absence of real systems in Section 4.2.

Model-based Anomaly Detection Chapter 5, then introduces a method for anomaly detection in cyber-physical system networks independent of the involved network protocols. Additionally, we provide a semi-automatic way for labeling training data and incorporate the detection method in a high-performance framework able coping with modern cyber-physical system installations.

Understanding Cause–Effect Relationships in Attack Campaigns In Chapter 6, we build a model for understanding cause–effect relationships in attack campaigns in larger networks. Using this model, we provide best practices for the deployment of intrusion detection systems.

Conclusion The last Chapter 7 adds final remarks to each of the previous summaries and points out future research possibilities.

2 Background

In this chapter, we introduce several concepts, definitions, and methods used throughout this thesis. As this work focuses on anomaly detection for cyber-physical systems, we start with definitions and details of these two concepts. After that, other methods used at various points in this thesis are explained.

2.1 Anomaly Detection

Term decomposition

While *detecting* something is a familiar concept for most people, the term *anomaly detection* depends on the meaning of normal and anomalous. Hence, for an understanding of the term anomaly detection, we decompose it into its foundational parts. In the field of IT security, a system operating as expected is usually considered being in the *normal* state. Once parts of the system deviate from their expected behavior, we face an *anomalous* state. The definition of expected system behavior is always specific to a certain use case and assumes some kind of reference model to which we compare actual behavior. There are many reasons why a system may behave anomalously. For example, possible reasons include:

Examples for anomalies

- degradation of system parts
- missed edge cases of *normal* operation during the design phase
- changes in a system's environment
  - changes in temperature, humidity, and electromagnetic or physical impacts (e.g., vibrations)
  - limited or no power supply
- unintended interactions by human users or other systems
- use cases of the system for purposes other those envisioned during their design phase

Other definitions

Other researchers [15, 16, 17], often consider intrusion detection being the overall topic. Here, they distinguish between signature-based and anomaly-based detection.

The signatures then usually refer to manually or semi-automatically derived rules for the detection of attacks, whereas anomaly-based methods incorporate some kind of machine learning. As basing a decision for a state being not normal based on manual or semi-automatically created rules still poses an anomaly, this distinction is not well-defined. To avoid this confusion of different meanings of the terms *anomaly* and *intrusion*, we clarify our usage of those in the next sections.

2.1.1 Anomalies vs Intrusion

In IT security, the terms *anomaly detection* and *intrusion detection* are often used interchangeably. However, there are differences between anomalies and intrusions, for sure.

While an

Anomaly is something not normal.

an

Intrusion is the result of an attacker tampering with the system.

These two definitions depend on the meaning of *normal*. In the previous section, we already introduced exemplary reasons for anomalies. Intrusions may, however, also result from problems corresponding to normal behavior. E.g.,

Examples for intrusions

- system crashes induced by an attack
- misuse by authorized people

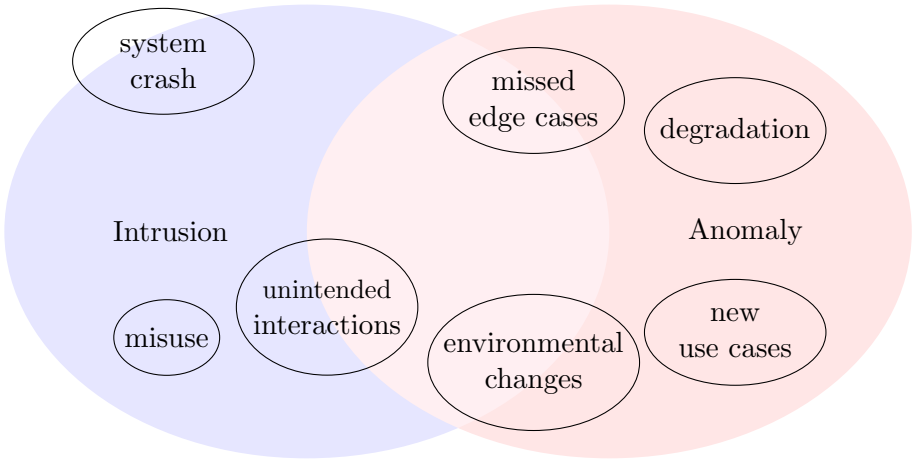

Figure 2.1: Exemplary clustering of different reasons for anomalies and intrusions.

Relationships between them

In the illustration in Figure 2.1, we see that intrusions and anomalies represent two different sets with a probably non-empty intersection [18].

For example, the crash of a system corresponds to normal behavior. In the case of corrupted input data, the system is supposed to crash and generate a corresponding log entry. However, an attacker may specifically craft input data to destroy a system's availability by continuously forcing it to crash. Some people declare a system crash as an anomaly. In real-world systems, we see system crashes from time to time during *normal* operation. Hence in this thesis, we consider system crashes as normal. Nonetheless, they can be the result of an intrusion.

This example explains why we prefer seeing anomaly and intrusion detection rather as two different tasks on the same abstraction level than subtasks of each other. Whether either of these tasks is solved using a method from the machine-learning domain, that does not change its purpose.

2.1.2 Detection

What is detection

When we talk about detection methods in this thesis, we consider a decision function

$$f_{deci} : X \to Y \tag{2.1}$$

where X represents the state space of the system, and Y represents the decision space for the detections.

However, the involved or used characteristics in a state $x \in X$ depend on the actual method used while the actual types of decision $y \in Y$ may be different as well.

The previous discussion of anomalies and intrusions already reveals that these two terms refer to decision functions operating with different output spaces Y. In both cases, there are two commonly distinguished variants: network-based and host-based detection methods.

Network-based detection

In network-based intrusion detection, the method uses data concerning the network as the input state x. Many business IT methods rely on using metadata of network packets, like MAC/IP addresses, TCP options, TCP sequence, and acknowledgment numbers. Then, x may look like

$$\vec{x} = \begin{pmatrix} mac \\ ip \\ seq \\ ack \\ synflag \\ ackflag \\ \vdots \end{pmatrix} \tag{2.2}$$

Other methods look at communication patterns, like average package sizes, communication partners, or the length of communication flows. In this case, $\vec{x}$ has different components correspondingly. Further, using deep packet inspection, specialized methods include features derived from the payload of the network packets.

Host-based detection

In host-based intrusion detection, data is gathered not on the network but from all information available at a single or multiple hosts. There we encounter, for example, information regarding running applications and their internal states.

Sometimes, researchers refer to application-level intrusion detection as a separate technology. During this thesis, we consider application-level IDS as host-based methods since they usually rely on information directly available at a single host. Apart from being highly specialized in just one application, they share the same properties as host-based systems.

Nature of f_{deci}

Finally, we can now understand what the distinction between methods based on signatures or machine-learning refers to. These describe the nature of f_{deci}, once we have settled for the two spaces X and Y. Machine-learning based methods, thus, usually yield us approximations of the decision function $\hat{f}_{deci} \sim f_{deci}$.

2.1.3 Differences to other Security Measures

Detection, no prevention

Anomaly and intrusion detection are by their definition non-preventing security measures. This means that they cannot prevent any attack on their own. Some IDS solutions come with an accompanying Intrusion Prevention System (IPS). These use identified anomalies to, e.g., subsequently block requests from identified malicious sources. In the domain of CPS, operators from industry usually fear the automatic interaction of active measures with their systems. As the availability of a CPS is usually considered being the top-most goal, a misclassification of an IPS may lead to even worse damages than a potential attack if the intervention is not specifically adapted to this use case. Still, the earlier a security incident is detected the more time remains for operators to actually react upon them and prevent possible large damages.

Approximations of f_{deci}

Additionally, if an IDS is based on machine-learning, it does not derive exact decisions but probabilities or confidence values for a specific decision. This fact is usually overlooked but nonetheless real for all learning-based systems. The only instance we might be 100% sure about a decision is if the current state has been encountered exactly the same during the learning phase. Otherwise, we always apply an approximation of the decision function to values we have not tested before.

2.2 Cyber-Physical Systems

What are CPS

While the previous explanations of anomalies and intrusion are general and not specific to a particular domain, we now introduce cyber-physical systems and how anomaly detection for these differs.

In this work, we follow the common definition of cyber-physical systems as a compound system where the physical word interacts with computation environments [1, 19, 20, 21]. Hence, a CPS is a mixture of both worlds characterized by interactions and influences along its interfaces. As stated by Lee [1], usually these interactions form feedback loops.

Examples for CPS

With this definition, we can give some examples of cyber-physical systems illustrating the generality of this definition and an idea of different domains this is applicable to.

Water Treatment Plants are facilities handling wastewater to clean it up again. In these systems, as representatives of the physical world, we find water and chemicals, the properties of the chemical cleaning process, and the environment itself as these plants are usually operated outdoors. In addition, pumps and valves, i.e., actuators, manipulate the ongoing process steered by control systems based on

respective sensor measurements. This describes an interaction through interfaces with computational methods.

Manufacturing Systems use manipulators (interfaces) to alter materials (physical) into the final product. With the developments of Industrial IoT, the whole process is supervised by sensors and then finally controlled (computation).

Power Plants similarly use IT-controlled processes to transform different types of energy into each other.

Home Automation controls the behavior of lights and window blinds based on respective sensors.

Automotive and Aerial Systems like autonomous cars and drones also represent a CPS according to this definition. These also present a controlled behavior based on sensor measurements.

While some of the examples are only small CPS, they still share the same properties of the examples described in more detail. The aim of these examples is not to be an exhaustive list but to go beyond the classic examples for CPS making aware that the methods presented in this thesis might be interesting for even more domains.

2.2.1 Differences to business IT

Assumptions about IDS

Anomaly and intrusion detection for CPS must answer the same decision function as in the business IT domain. The main difference to general business IT methods resides in the characteristics of our input space X. IDS deployed for business IT security rely on several assumptions. The well-known Network Intrusion Detection System (NIDS) `snort` [22], `bro` [23], and `suricata` [24] all rely on parsing network packets of known network protocols. Additionally, they incorporate off-site analysis of the network traffic encountered as the processing is resource-intensive. The same is true for the Host Intrusion Detection System (HIDS) `OSSEC` [25] and `wazuh` [26] which require access to local files on the hosts and an off-site analysis systems. If such capabilities are not already presented in a particular device, it is obviously needed to change the system to allow the execution of these IDS. Hence, in IDS for business IT, we have specific assumptions about X. For network-based systems, we assume:

A1 We generally see open and known network protocols as they must be understandable for a heterogeneous set of clients.

A2 Hence, the network packets and their payloads are easily parseable.

A3 Network speeds are fast to allow for easy transmission to off-site analysis systems.

A4 We can change network infrastructure to enable data acquisition for anomaly detection.

For host-based systems:

A5 We can easily access data residing on the hosts.

A6 We may change systems to acquire information.

Violated assumptions in CPS

All of these assumptions cannot be hold up in CPS. At least to some degree, existing constraints prevent the application of those assumptions. CPS networks usually face closed-source and vendor-specific network protocols that cannot be parsed except for specialized clients (A1, A2) [27]. While some CPS networks deploy similar technologies as business IT networks, especially Fieldbus networks have way lower bandwidths. Therefore, the amount of transferable data is constrained and usually cannot be increased for non-functional purposes (A3) [27]. Restrictions like power and processing limitations, safety certifications, and closed-source components prevent many needed changes to incorporate security-specific sensors or systems (A4) [7]. This also usually prevents us from adding information gathering components into existing host systems (A6) [7]. Being closed-source systems, even the access to already existing information is often not documented and thus not possible (A5) [27].

2.3 Machine Learning

Classification vs. regression

For the purpose of anomaly and intrusion detection methods, we will rely on a few general concepts from the machine learning domain. In general, we divide machine learning methods into classification and regression methods [28]. Referring to our decision function (Equation 2.1), *classification* regards Y as a discrete set of values, e.g. *normal* and *malicious*, whereas *regression* rather resembles a continuous space Y, e.g. $(0, 1)$.

Supervised vs unsupervised

Further, we can distinguish different machine learning methods regarding their learning strategy. There are *supervised* and *unsupervised* methods [29]. While a supervised method requires exemplary points of our decision function f_{deci}, unsupervised ones only require a set of points of the input space X. Hence, a supervised method requires us to answer the decision function for some examples in advance before applying our actual

method. These well-known examples are generally referred to as *ground truth*. In the case of machine learning methods for IT security, we usually do not have a ground truth as data from real attacks is not public, or has uncertainties attached. Thus in this thesis, we will mainly stick to unsupervised classification methods. In particular, we will use a variant of deep learning approaches called stacked denoising autoencoders. The details for that method are outlined in Chapter 5.1.2.

2.4 Related Work

As we have discussed the needed background for this work, we now introduce related work concerning our contributions.

2.4.1 Attack Landscape for Cyber-physical Systems

Bridging the gap between virtual and physical worlds, CPSs introduce several new attack vectors. Antón et al. [2] give a summary of current exploits and trends in attacks on CPS. Particularly, they state that exploitation of industrial equipment like Programmable Logic Controllers (PLCs) is becoming more relevant while many CPSs still do not implement basic security principles. Anomaly and intrusion detection systems try to find security incidents as early as possible to enable fast recovery. This is even more important with the high damage potential of CPS. Therefore, we pay particular attention to such attacks. Recent attacks such as Stuxnet and Duqu [30] showed damage potentials ranging from theft of intellectual property to physical damage. These attacks have been analyzed in detail in the past [3, 31]. The reports show that attackers should be assumed to be very knowledgeable about the attacked systems. Attackers effectively manipulated application logic of Supervisory Control and Data Acquisition (SCADA) systems, which developers believe to be too specialized to be understood by someone not involved in its creation. Other analyses of similar attacks like Duqu, Flame, and the attack on the Ukrainian power grid came to the same conclusion [4, 5]. Therefore, although CPS face the same threats as other systems, there are also yet unknown and targeted attacks for specific systems.

Turner et al. [32] showed that among mechanical engineering students the awareness of such problems is low. The students were not able to link problems with the quality of the resulting products to security-related problems.

Possible reasons for this increase in attacks on CPSs can be found in [7]. In the study, Fernandes et al. [7] conclude that CPSs have specific properties differentiating them from classical business IT. Among others, they mention limited resources, communica-

tion protocol diversity and repurposing of known technologies in unforeseen ways. They conclude that, hence, some of the established security measures cannot be transferred directly to CPSs. Other security practices, however, may benefit from the well-defined behavior of a CPS which should ease the learning of system models, eg. for anomaly detection.

Fernandes et al. [7], only mention one approach based on fingerprinting. In our approach, we take advantage of that well-defined behavior to train a stacked denoising autoencoder enabling anomaly detection in even unknown protocols.

After these considerations of attacks with high damage potential on CPS, we assume to face an attacker as outlined in the Dolev-Yao model [33]. However, we are aiming for an early detection of attacks not the actually prevention of them. Therefore, we need more details on what types of attack we can detect than on whether an attacker is able to execute them. We give such details along with the described methods in the following chapters.

2.4.2 Data Acquisition for Anomaly Detection in CPS

To apply anomaly detection to any use case scenario, it is a pre-requisite to obtain suitable data. There are basically three different possibilities to acquire such data: direct acquisition, generation, or use of published data.

2.4.2.1 Direct Acquisition

While network capturing is a well-established technique, published datasets (cf. Table 4.1 and Section 5.2.3) show that anomaly detection in industrial settings may see large amounts of data. For a bandwidth-friendly acquisition of this data, reduction and compression techniques such as the *Deflate* algorithm of the Zlib library [34] can be used. However, common compression algorithms like LZ77 [35] in Zlib only use a fixed window size in which data is replaced. Redundancies that occur on a lower frequency than this fixed capture window cannot be eliminated. Additionally, as this algorithm uses a relative referencing approach instead of a direct referencing, it is required that the data is decompressed in the same order as it was compressed.

The approach which we present in Chapter 4.1, instead, relies on the fact that network traffic in CPSs may repeat with arbitrary frequencies. Also, we allow for decompression of the data in an arbitrary order. This effectively enables parallelization and offloading to multiple worker nodes after the data acquisition as is described and utilized in Section 5.1.3.

2.4.2.2 Generation

Generating data for anomaly detection in industrial settings has previously been done for the creation of fixed data sets. However, as the goal was to produce a fixed data set, the methods used usually do not allow for easy adaptation to new use cases.

Later, Lemay and Fernandez [36] introduced a first Modbus dataset that originates from simulations. Their system is based on self-developed programs to generate the data. For their dataset, they simulated a power grid system and used that data in a fixed topology to derive a Modbus dataset.

Current simulations

Instead of using a real scenario, [15] describes a simulation framework to verify the security state of a networked water level control system. To measure the performance of their detection approach [37] use a custom simulation testbed which is not described in detail. By using a handcrafted feature set they detect intrusions in Modbus traffic. However, all these approaches are specific to each use cases and cannot directly be applied to others.

Existing simulations which allow for easier adaptation have been presented by [38, 39]. Antonioli and Tippenhauer [39] present a toolkit which can be used to reimplement the networking parts of CPS installations. Their tool *MiniCPS* is able to recreate different network topologies and specific industrial network protocols. However, their toolkit does not include means to recreate realistic data to be transmitted over the network. They assume that meaningful values are already available. In contrast, Chabukswar et al. [38] also simulate the physical layer but as their network simulation relies on the OMNET++ framework it lacks the possibility of simulating more complex protocols. Our approach in Section 4.2 will combine the two ideas, simulating the network as well as the physical layer while still allowing adaptation in both layers.

2.4.2.3 Datasets and Testbeds

A viable source of recorded and documented network intrusion test data gathered from enterprise networks are the DARPA challenge datasets from 1998 to 2000 [40]. They contain conventional office IT traffic as well as some well-described attacks but no CPS-specific communication.

There are few datasets that can be used to compare the performance of intrusion detection systems on industrial network data. A more recent, CPS-specific Modbus dataset, including attacks and labels, is provided by [41]. These have been generated by a simulated control system, but are still interesting for first analysis. Another dataset covers the traffic of an industrial test lab for hands-on testing captured during

the 4SICS conference in Sweden [42]. Although it is quite big, this dataset does not contain any labels or information about possibly included attacks.

Surveys

In [43], about 30 Industrial Control System (ICS) testbeds have been reviewed. However, the authors found that less than half of them actually tried to verify the acquired data. Candell Jr. et al. [44] show one example of an ICS testbed where the authors evaluate different processes. As they rebuild the actual network infrastructure of their specific application, their approach is not directly transferable to other use cases.

McLaughlin et al. [45] found that while ICS testbeds usually address vulnerabilities in one layer, e.g., the field devices, attacks most often target several layers. Therefore, they argue that multilayered ICS testbeds are needed to analyze the vulnerabilities and develop countermeasures effectively.

Our framework provides such a multilayered approach by merging two existing frameworks into one (cf. Section 4.2). Further, we do not restrict our framework to one specific application but rather describe how operators of CPSs can use the framework to model their own use case.

2.4.3 Anomaly Detection in CPS

At the moment, current research is investigating two different approaches to anomaly detection in industrial settings. The first tries to adapt business network anomaly detection to the Fieldbus protocols used in the industry. The second approach includes more domain-specific knowledge into the detection methods to find anomalies based on an altered behavior of the industrial process.

Metadata-based detection

The authors of [46] use discrete-time Markov chains to detect alterations in sequences of network communication of a CPS. As they tested their approach on real-world data of a real plant, the effectiveness of their approach cannot be directly compared to other solutions. Often, telemetry analysis is used to detect anomalies in industrial network traffic [37, 47]. However, it has been shown that detection purely based on metadata is not sufficient for industrial contexts. [48, 49, 50] were able to demonstrate successful manipulations of process logic which are undetectable by metadata-based anomaly detection methods.

Process-based detection

Proposing a clustering approach on the actual process data, the authors of [51] chose a different method to detect anomalies in industrial process data with an accuracy of up to 98%. Several articles already concluded that the integration of process data into intrusion detection systems leads to an increase in accuracy. According to Turner et al. [32] and Pasqualetti et al. [52], neglecting these properties results in inferior attack detection.

Additionally, further approaches use a thorough analysis of the CPS to judge the behavior using either mathematical models [53] or simulated data [54]. While Haller and Genge [53] construct a mathematical model of the underlying process and exercise it on a chemical process, Potluri et al. [54] use simulated data from a robot arm to detect data injections. With the digitization of industrial processes, companies also start to build complete digital twins for the system, processes, and their interactions. In [55], such a digitalized twin of the system allow for comparing current real states with expected ones. Also, the input and output to processes and algorithms in industrial automation scenarios allows for detecting anomalous states [56].

Machine-Learning based detection

While applying machine learning to network anomaly detection is not a new idea, current methods do not take advantage of the characteristic properties of CPSs mentioned before.

Beaver et al. [57] evaluated different anomaly detection methods based on Support Vector Machines, Decision Trees, Random Forests, and Nearest Neighbors for CPSs. Despite good performance, they conclude there is still a need for improved features as input. In contrast to our approach in Chapter 5, they extracted protocol-specific fields of the network packet payloads to analyze the traffic. Hence, their method is bound to a specific protocol and may suffer performance problems as described in Chapter 4.1.2.

An application of an ensemble of autoencoders for network anomaly detection was presented with the `Kitsune` framework [58]. While the authors use autoencoders for traffic classification, the framework includes a separated feature extraction step, which requires at least a basic understanding of the involved network protocols. In their framework, Mirsky et al. [58] also focus on an efficient online application on low resource devices by using a Raspberry Pi. However, compared to control equipment in CPSs, also a Raspberry Pi may be considered a powerful device.

Feature learning, however, has already been applied in the domain of image recognition [59].

Apart from methods analyzing the network traffic, there are scenario-adapted anomaly detection approaches that require the modeling and understanding of the underlying CPS [53]. Similarly, Potluri et al. [54] use data from a simulation to detect false data injection attacks in a production setting. Instead of assuming that identifying sensor and actuator values from network packets is achievable, we provide a feature learning approach that extracts the required information automatically from network packets captured in the CPS.

2.4.4 Performance Evaluation of IDS

While many anomaly detection methods have been proposed, there is less research on the usefulness of their outputs and how their benefit can be maximized.

2.4.4.1 Alert and Log Aggregation

While data and log aggregation are in widespread use for big data applications, the security community currently adopts using Security Information and Event Management (SIEM) as well as logging systems. An overview of existing systems is given in [60]. In contrast to these systems, our framework extends to a semi-automatic acquisition of information. Additionally, we provide a method based on the collected data to assist the understanding of attack steps and their current progress. Research suggests that understanding the lateral movement of threats throughout a network is inevitable for its mitigation [61]. Additionally, concepts for network data aggregation are already available. Sadighian et al. [62] propose a model for an ontology-based alert fusion similar to ours. While they integrate several network intrusion detection sensors, they do not consider missed alerts and tracing the attacks throughout the network. For low resource anomaly detection under constraints, Sedjelmaci et al. [63] show a game-theoretic model and simulation to derive theoretical characteristics.

In all cases, task-specific data is vital for the effectiveness of the solutions. While data from different abstraction layers have proven to be efficient for anomaly detection [10], even in their absence, simulations represent a reliable source for information retrieval and data acquisition, as shown in Chapter 4. Additionally, we now see more distributed attacks [64] requiring us to join local IDS into more capable distributed setups.

Recently, the industry adopts distributed and highly connected network setups [65, 66, 67, 68]. While distributed anomaly detection is well studied for classical business IT [69, 70], there is yet little information for CPS [71]. Karimi et al. [69] use an Apache Spark cluster and the Hadoop Distributed File System (HDFS) for aggregating network traffic and a consolidated analysis. Igbe et al. [70], instead, analyze the traffic independently on each system and use the results of other systems as input for their own analysis. However, for CPSs with their decentralized setups, different aggregation mechanisms are needed. Thus in Chapter 6, we show how to aggregate information from distributed anomaly detection systems in CPSs.

2.4.4.2 Optimization

Furthermore, an optimized placement of available IDSs nodes may increase the overall system performance.

Chen et al. [72] present a first technique to optimize IDS sensor placements with genetic algorithms and suggesting suitable and intuitive resulting places for the IDS. More formal approaches use attack graphs and model the question as a minimum set cover problem, which is known to be NP-hard [73]. For safety sensors, Isovitsch and Van-Briesen [74] investigated optimal quality sensor placement using geographic information and statistical correlation features. However, they focus on physical manipulations of the water quality instead of IT security attacks.

In contrast, we define a simulation model enabling the search for optimal placements of arbitrary anomaly detection systems in different network architectures even under placement constraints (cf. Section 6.2).

3 Anomaly Detection Process

From SRA to Requirements

Currently, available standards for IT security in industrial use cases like IEC62443 [75] usually start the process of securing systems by identifying the highest risks. For this purpose, they consult Security and Risk Assessment (SRA), e.g., [12, 76], to derive security requirements that need to be met by the system. A SRA assesses the damage potential of violated security goals of every asset in a system. In combination with threats and their corresponding attack potential, risks for the system can be determined. For the mitigation of these risks, security controls are introduced. Often, security concepts for industrial use cases cannot utilize the same security controls as in business IT for the respective requirements. Long life-cycles of used machines and expensive safety certifications may impair the deployment of specific security measures. A less invasive approach to address regulatory compliance is often to deploy anomaly detection allowing for at least being aware of possible attacks [60]. For this purpose, we need specialized IDS able to work on the already available data to avoid the described restrictions. As this is usually the first point of contact for CPS operators with anomaly and intrusion detection, we start this thesis by analyzing different security requirements and transferring them to anomaly detection. While some security measures directly refer to an anomaly detection approach, for others, only their fulfillment can be monitored. Although monitoring is often regarded as a security measure itself, we use it in this chapter as a trigger for anomaly detection. In fact, a monitoring system reporting the non-fulfillment of a requirement is itself a form of anomaly detection.

Requirements to Anomaly Detection

Contribution of Chapter

This analysis shall also deepen the understanding of the various forms anomaly and intrusion detection can appear as. Finally, we bring all these different aspects together and introduce a modular architecture able to integrate all the aspects and allowing for a use-case specific adaptation.

Parts of this chapter have previously been published in “Deriving Impact-driven Security Requirements and Monitoring Measure for Industrial IoT” by Gerhard Hansch, Peter Schneider, and Gerd Brost [12].

Structure of chapter

This chapter first presents a security toolbox in the form of a catalog for the derivation of intrusion detection and system monitoring requirements based on SRAs tailored

for the IIoT and CPS domain. By mapping different security measures to respective requirements, it allows for deriving rules for current anomaly and intrusion detection solutions. Enabled by this mapping, the catalog supports the often laborious definition and prioritization of monitoring rules by an SRA-based automated approach. By connecting the catalog with an SRA, we provide a framework that dynamically derives recommendations and requirements from a variety of protection and monitoring measures and techniques. Thereby, we provide a general methodology that helps operators to identify, justify, and implement suitable measures in order to strengthen the overall security of their systems.

3.1 Protection and Validation

Model for requirements

With security and risk assessments, we derive security requirements based on security objectives and different asset classes. For IIoT and CPS, relevant security objectives are *confidentiality*, *integrity*, *authenticity*, and *availability*. On an abstract level, assets can be grouped into system *component*s carrying out a *function* which operates on or generates *data*. Communication between functions and components is possible by data transmission over *connection*s. These two concepts allow for categorization of assets in these classes and an attachment of the required security objectives. Tables 3.1 and 3.2 then provide a catalog for protective security measures and monitoring and intrusion detection solutions.

Objective	**Function**	**Component**	**Connection**	**Data**
Confidentiality	Obfuscation, Runtime protection	Access control, Storage encryption	Encryption	Encryption
Integrity	Runtime protection	Trusted/Secure boot, Runtime protection	Authentication, Firewall	Cryptographic Hash
Authenticity	Signature	Digital identity	Access control	Signature
Availability	Error handling, Redundancy	Redundancy	Redundancy, Error handling	Fallback values

Table 3.1: Protection measures per security objective and class.

Objective	Function	Component	Connection	Data
Confidentiality	Memory and output analysis	—	Cryptanalysis	Cryptanalysis
Integrity	Software test, Comparison	Comparison, Trusted/Secure boot	Traffic analysis	Comparison
Authenticity	Functional test	Trusted boot	Second channel comparison	Signature check
Availability	Heartbeat, Timeout	Link detection	Link detection	File access

Table 3.2: Validation measures per security objective and class.

There are many methods for SRA already available [12, 77, 78, 79]. They all provide a way of identifying most relevant security problems and derive further requirements to lower remaining risks. For each requirement defined in such a SRA, a suitable protection measure is available. Depending on the modeled asset category and the security objective to be protected, Table 3.1 provides a mapping for applicable security measures. For each of them, the following paragraphs give details on how they increase the security posture. Additionally, we suggest measures to validate the fulfillment of these measures automatically. By this, monitoring the effectiveness of security measures can become a form of anomaly detection. However, as our summary in Table 3.2 shows, this is not yet possible for all the security measures. Especially in the case of availability of assets, a violation of the security objective can have other reasons than a direct attack. Therefore, the following paragraphs give short explanations for available protection measures and how their fulfillment can be validated regarding a use as an indicator for anomalies.

Mapping to protection and monitoring

3.1.1 Cryptographic Hashes

A cryptographic hash uses a one-way function to calculate a unique identification hash for a given byte sequence. Relying on robust functions, they can be used to generate identifiers which do not allow to derive the original byte sequence based on the hash

result. Thus, they can be used as a unique identification of data without revealing their contents.

Validation Assuming that the content of some data artifact is known, hashes can be calculated on both sides. A comparison of the resulting hash is enough to check the validity of a huge amount of data. However, it is worth noting that so far some earlier hash functions have already been shown to cause collisions and, thus, should not be used anymore [80].

3.1.2 Signatures and Digital Identities

While cryptographic signatures are a great tool to provide authenticity for information, they usually require the setup of a Public Key Infrastructure (PKI) in IIoT use cases [81]. If we face a requirement indicating the need for signatures, it is, thus, needed anyway, that all affected components have access to methods to verify the signature.

Validation The same method can then be used by a validation system. If the method is not available, not able to verify a signature, or there is no signature for the required data or function available, an alert must be logged.

For data, there is an alternative validation method based on the idea of digital twins. Data transported in IIoT use cases is consumed or produced by some function. Thus, data relates to general input and output in software engineering. A possible solution for monitoring the validity of such inputs and outputs is, therefore, applying the same measures. Recent work [56] showed the effectiveness of checking the input validity to functions in industrial use cases. However, these methods must be adapted to each data type and function.

3.1.3 Runtime Protection

McLaughlin et al. [45] suggest several software mitigation strategies specific for embedded and Cyber-Physical Systems. Among these are control flow integrity and code randomization aiming to provide a secure execution. Specific architectures like Intel's Trusted Execution Technology (TXT) and ARM's TrustZone represent a suitable measure to execute code in a restricted enclave, effectively sandboxing it from other code on the same processor. Thus, the restriction of shared hardware resources and information transfers prevents many side-channel attacks. The restricted access ensured by corporate production environments similarly yields a physical runtime protection in IIoT scenarios.

Validation We can only verify the integrity of proprietary and vendor-specific sandboxes if a suitable checking mechanism is provided. Regular penetration tests and security audits may reveal possible vulnerabilities. However, manipulations may still occur even in air-gapped systems, e.g., by social engineering attacks.

3.1.4 Trusted Boot

Among others, trusted boot can be implemented with Trusted Platform Modules (TPMs). These measure the executed code in a physically secured environment. By implementing trusted boot, a TPM can provide hashes of executed software beginning with the boot phase up to software executed in userspace. Any modification to the initial setup of a component can, thus, be detected. Apart from this component integrity measurement, TPMs also provide capabilities for storing identification material such as private key material of certificates in a physically secured space.

Validation The measured hashes during the boot phase can be used to verify the running software. In a trusted boot strategy, the next software component verifies the measured hashes with predefined ones. Thus, a trusted boot mechanism already validates itself.

3.1.5 Encryption

In industrial network protocols, there are only limited encryption capabilities. However, recent research developed new methods for different protocols [81, 82, 83, 84]. Those are usually based on wrapping the Fieldbus communication in TLS tunnels or slight modifications of the protocol itself.

Validation Using cryptanalysis methods, weak encryption ciphers can be identified. As these methods try to break the encryption without the proper key material, they can only identify non-working setups. Thus, the lack of a corresponding method does not validate the efficiency of an encryption method. Hence, we rely on heuristics to validate the encryption's efficiency (see Section 3.2.4).

3.1.6 Redundancy

Using redundant systems, the availability of components and connections can be secured. In the case of industrial field buses, like CAN [85], this can be achieved with redundant physical connections.

Validation The redundant components and connections can be validated by using explicitly specific ones. Custom test cases for this check can be developed.

3.1.7 Firewall

Firewalls act as a gateway for ingoing and outgoing data at a specific point. The definition of predefined rules for metadata, content, or connection state enables fine-grained control. However, a detailed configuration quickly gets rather complex. By filtering the data flow, a firewall, as proposed in [86], enhances the integrity of all passing connections.

Validation By systematic sending of forbidden data or along constrained connections, we can validate the firewall rules and test their effectiveness. This is essentially analog to unit testing the firewall rules. Therefore, specific tests must be developed for every deployed setup.

3.2 Continuous Monitoring the System

Using a lightweight security analysis, we derive a set of requirements. These security requirements and their corresponding protection measures can now be translated automatically into rules for a continuous monitoring system. For each rule, we define an alert level which defines the severity of its violation. Here, the damage potential of the violation of any security goal can define this alert level. While the methods presented so far are diverse, they all build upon the same technical requirements as they tackle a decision function — measure fulfilled or not (Eq. 2.1)— in different ways. A suitable monitoring and detection system therefore includes: data acquisition, data preprocessing, data forwarding, data analysis, and data reporting.

Identifying suitable solutions for data acquisition and preprocessing, additionally, requires an understanding of the different available data types which are relevant in IIoT scenarios.

3.2.1 Data Categorization

In the field of IIoT, there are several data sources that can be monitored. Table 3.3 lists examples for data sources. These data sources can be categorized according to their *location* where the data can be recorded, its *data type*, and the place where its *analysis* should take place. Most data are best recorded locally at each machine participating in the system. Recording locally ensures later analyses are based on

Local data

the actual data as it arrived or has been measured reducing the risk of intermittent modification. However, as the evaluation of measured data requires significant resources and may depend on other data, remote systems provide interfaces for analysis. While some intrusion detection frameworks already incorporate such distributed setups, a more generic approach for distributed data acquisition in the case of network traffic is given in [8]. In stages before the final deployment of the system, the output of simulations such as described in Chapter 4.2 gives a basis for the implementation and test of suitable analysis mechanisms.

Remote data

Alternatives

Data source	Location	Data type	Analysis
executed software	local	measurement	remote
application data	local	parse-able format	local
log data	local	text format	remote
network architecture	global	metadata information	remote
network behavior	local	measurement	remote
network traffic	local	measurement	remote
system behavior	global	measurement	remote

Table 3.3: Data sources relevant in IIoT settings and corresponding monitoring techniques.

3.2.2 Data Acquisition and Preprocessing

As pointed out previously, most data available in IIoT scenarios is best measured locally but analyzed at a remote location. Since these scenarios incorporate a lot of different components, an efficient strategy for preprocessing data is required. Considering the different data types available, a set of different processing methods is required. While system log files can be transmitted directly, network traffic may be compressed and forwarded using secured connections as described in Section 4.1.

Preprocessing

In an ideal system, application data which is directly involved in the IIoT process should be subject to an input and output validation [56]. Logs produced by this validation procedure should also be forwarded to an offsite analysis system. The executed software on each system can be measured using cryptographic hashes. Further, using a hardware trust-anchor and measured boot concepts on the systems, these hashes are protected from manipulation. As the network architecture and system behavior are global properties of the systems, no preprocessing can be done on them but instead the measurement should be considered.

Recording

3.2.3 Data Analysis and Reporting

Forwarding

After preprocessing the data, we can transmit it to an offsite analysis system using secured connections, e.g., by the use of mutual TLS. To ensure a trusted communication, TLS with client and server authentication is required. Depending on the complexity of the network, pre-shared device-specific keys may be used for the authentication. In networks with higher system complexity or dynamics, the use of client and server certificates should be considered.

Analysis and reporting

The analysis system is in charge of checking all defined validation methods or rules against the data provided by the participating systems. Violated rules lead to an alert and are signaled to the system operator in a SIEM system. Additionally, the SIEM allows for manual checks of all provided data. Suitable filtering and correlation techniques may be used for advanced manual analyses.

The core of the detection of anomalies and intrusions is, thus, the definition of suitable rules for the data provided.

3.2.4 Monitoring Rules

Table 3.4 lists monitoring measures to supervise the fulfillment of the derived security requirement implementations as well as validate ingoing and outgoing data. In the following, for each of these measures, details and examples are given to derive corresponding monitoring rules automatically.

Objective	Function	Component	Connection	Data
Confidentiality	Runtime monitoring	Tamper detection	IDS	Entropy checks
Integrity	Integrity monitoring	Tamper detection	Network monitoring	Plausibility checks
Authenticity	Fingerprinting	Identity validation with HSM	Network monitoring	
Availability	Runtime monitoring	System monitoring	Network monitoring	

Table 3.4: Exemplary Monitoring measures per security objective and class.

```
<syscheck>
    <directories report_changes="yes"
        realtime="yes" check_all="yes">
        /path/to/folder/or/file
    </directories>
</syscheck>
```

Listing 3.1: Declaration of integrity scans for files and folders in the free IDSs OSSEC [25] and wazuh [26].

```
INSERT INTO files (file, md5sum)
VALUES ('/path/to/file', '22ee5cebb5bddfad5490633dab7d1afc');
```

Listing 3.2: Whitelisting known hashes of files for integrity scans using OSSEC.

Tamper Detection Checking the file integrity on affected components guarantees the detection on any unintended changes. A rule for state-of-the-art host-based intrusion detection systems to check the integrity of a specific file or folder, e.g., a software artifact, or some data asset, is shown in Listing 3.1.

Such a configuration is used locally to derive a list of files and folders to compute hashes on a regular basis or upon changes. As many systems in IIoT scenarios are based on the Linux operating system, a real-time monitoring can be deployed using the `inotify`-subsystem [87]. A whitelisting approach of specific hashes allows for trusted changes in the measured files. Whitelisting hashes of known and trusted files can be done by adding them to a database (cf. Listing 3.2). With wazuh [26] and OSSEC [25], there are two host-based detection frameworks available with inbuilt whitelist support. While these systems monitor file changes using `md5` and `sha1` hashes, the whitelists only support `md5` hashes, which may be considered too weak [80]. Therefore, custom implementations of such logic shall use state-of-the-art cryptographic hash functions.

Reducing false positives

By comparing new hashes against those stored in a whitelist database, the offsite analysis system can suppress false positive alerts for intended file changes. A corresponding configuration option for OSSEC/wazuh is shown in Listing 3.3.

Network Monitoring The huge amount of network traffic arising in IIoT scenarios may either be analyzed locally or remotely. In case the device has enough resources, local processing using state-of-the-art NIDS, e.g., snort [22] or suricata [88], ensures immediate detection of threats. However, most IIoT scenarios are built upon low-

```
1 <global>
2     <md5db>/path/to/md5.db</md5db>
3 </global>
```

Listing 3.3: Defining a whitelist database for integrity scans using OSSEC.

resource devices which profit from a remote analysis. In that case, local acquisition, lightweight compression, and forwarding, as shown in [8], minimize the impact of the monitoring as much as possible. After transmission, state-of-the-art NIDS perform the analysis on batches of the recorded data.

Service enumeration

We define the network behavior for each device as the set of all services and listening ports and addresses of it. Most operating systems provide a suitable mechanism to derive such a list using onboard tools. For Linux, such a list may be derived as in Listing 3.4.

```
1 ss -tlp
```

Listing 3.4: Listing all processes for TCP connections with their ports on a Linux machine.

The output of these tools should be logged locally and reported to the offsite analysis system.

Global view

The analysis system must, additionally, monitor the network for its architecture. This includes details about communication partners and paths. As IIoT use cases always incorporate physical sensors and actuators to some degree, a feedback loop between the physical world and its digital counterparts is given. This feedback is at least in parts shared over the network for digital control of the underlying physical processes. Therefore, a suitable system for tracking and analyzing the system behavior is needed. Such systems have been previously described in [10, 46, 52, 53, 54, 89].

Link Detection The availability and configuration of network interfaces can be detected using onboard tools on almost all common operating systems.

On Linux available and configured network interfaces may be retrieved as in Listing 3.5.

```
1 ip addr | grep "state UP"
```

Listing 3.5: Listing all active network interfaces on a Linux machine.

Heartbeat/Timeout If a specific service is up and running can be checked locally as shown in Listing 3.6.

```
1 ss -tup | grep LISTEN
```

Listing 3.6: Listing all local services on a Linux machine.

From remote systems, testing the availability of a service is best done by a connection attempt. A failed connection attempt indicates an unresponsive or unreachable service and must be logged. For this we can use port scanners like `nmap` (cf. Listing 3.7).

```
1 nmap -sS 192.168.178.1
```

Listing 3.7: Listing available services from remote.

Functional Testing If a function must be treated as a black box, testing the behavior of the function indicates whether it is working as intended or not. Such a functional testing can be implemented with comparatively low effort using unit tests from software testing. However, specifically tailored tests may produce even more accurate results.

Entropy of Encrypted Storage and Data While the encryption of transferred data and storage affects two different classes in the previously described catalog, the measure to protect their confidentiality is the same. As encryption of data and storage essentially provides a Binary Large Object (Blob) that cannot be checked for its validity, it is hard to monitor if such a requirement is met. However, we can measure the entropy in the binary blob by reading the file or storage medium and calculating the entropy given as

$$E = -\sum_{\alpha \in B} p_\alpha \log_2(p_\alpha) \tag{3.1}$$

whereas B denotes the set of all possible byte values, i.e., $B = \{0x00, 0x01, \ldots, 0xff\}$ and p_α represents the frequency of occurrence of α in the investigated storage medium. Assuming data is encrypted using state-of-the-art algorithms, the frequency of occurrence from different bytes must be almost identical. If this was not the case, then the encryption algorithm either did not manage to hide the information, which would allow for attacks, or the data was not encrypted at all. Assuming a proper encryption has taken place, for every $\alpha, \beta \in B$ the frequency should be similar, i.e., $p_\alpha \approx p_\beta$.

```
1 ps aux
```

Listing 3.8: Listing all running processes on a Linux machine.

Hence, for a theoretical correct encryption Equation 3.1 simplifies to

$$E^{enc} = \log_2(|B|) \tag{3.2}$$

whereas in all other cases, the entropy E will be less than E^{enc}.

Thus, the measurement of entropy in a file or storage medium yields indicators for how well encryption has been implemented and applied to it.

A simple implementation of this entropy check can be obtained using compression methods. Similarly to the compression method outlined in Chapter 4.1, after capturing and compressing network traffic, the compression rate gives an estimate on how much a data stream is encrypted. We show that unencrypted industrial network traffic can be compressed up to 64%. This means it carries much redundant information that relates to low entropy.

Fingerprinting Similarly to the whitelisting of known executables and files discussed before, we can calculate fingerprints of important files. Using hash-functions, we obtain a fingerprint of the current state of every file. A simple monitoring procedure can now check whether any of the monitored files were ever modified by recalculating hashes and comparing them to the history.

3.2.5 System Monitoring

Continuous logging of all running processes enables the identification of newly spawned processes (cf. Listing 3.8).

3.3 Monitoring Architecture

Finally, the described process of data acquisition, preprocessing, analysis, and reporting requires a flexible architecture, as shown in Figure 3.1. There is no need to require a particular network topology as long as each component may reach the offsite analysis system. In Figure 3.1, we demonstrate how such a continuous monitoring system can be implemented. The processing starts from a set of data sources, $src_1 src_2, \ldots, src_n$

in an input layer. From there, acquired data can be directly forwarded to the preprocessing stage. To enable the acquisition from large CPSs, compression methods as described in Section 4.1 can be used. The preprocessing stage then transforms the data into the filtered representations $rep_1, rep_2, \ldots, rep_m$. Afterward, the different monitoring mechanisms outlined in Section 3.2, $M_1, M_2, \ldots, M_o$, use each a subset of the representations generating alerts at corresponding alert levels at the earliest moment possible. A scheduler uses already available alerts and supplementary data representations to issue further investigations in dedicated systems. This includes comparing data to expected data from digitally twinned systems [55], checking identified system invariants [53], or issuing a manual analysis request to a system operator.

3.3.1 Preprocessing Stage

Data source	**Preprocessing**
executed software	hashes of binaries
application data	data validation logs
log data	whole log files
network architecture	*not applicable*
network behavior	listening addresses/ports and associated processes
network traffic	compression and forwarding
system behavior	*not applicable*

Table 3.5: Local preprocessing of data sources to facilitate and streamline the detection approaches.

In the preprocessing stage, data for each element of the CPS must be collected and converted to a suitable representation (cf. Table 3.5). In Table 3.3, we already listed possible data sources. For an actual implementation, however, we need to distinguish classical IT systems and interfaces to the physical world: IT systems can deliver a lot of data such as executed software, application and log data, and network traffic. As sensors and actuators, i.e., interfaces, are small embedded and highly purposed devices, they do not directly deliver data. However, firmware updates sent via the network, as well as control and measurement signals transmitted may be used as input data. Accessing data from the physical world is hardest as we can only rely on dedicated sensors. An analysis of the system behavior gives insights on what process values to observe. A data historian, as used in some SCADA systems, also offers a wealth of sensor, actuator, and process data.

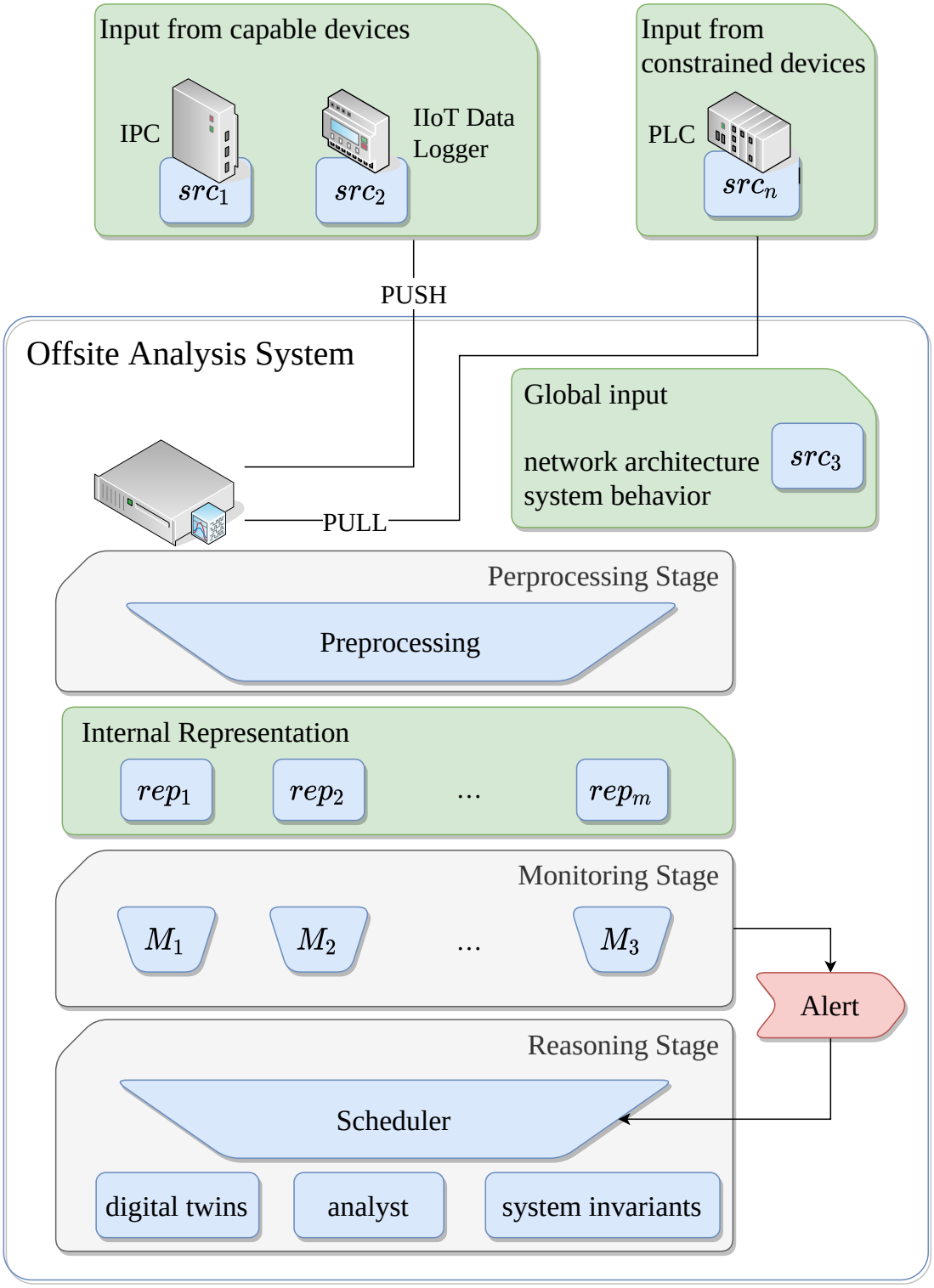

Figure 3.1: Architecture for holistic monitoring and intrusion detection.

3.3.2 Monitoring Stage

In the monitoring stage, we can now use a collection of the derived monitoring measures as well as latest anomaly detection methods. While our method from Chapter 5 can work on network data from a large set of protocols, we are aware that specific scenarios might already have more accurate methods available. Therefore, we assume to have a set of methods to be applied in the monitoring stage. While using different algorithms, all these methods use data as described before to generate alerts. Applying them in a streaming pipeline leads to the association of new signals with already known data points. By that, they have fast detection performance with varying accuracy. Making the root cause of any detection transparent to a CPS operator, however, requires further processing.

3.3.3 Reasoning Stage

A scheduler forwards generated alerts to either reasoning-based systems or a manual analysis by an expert for that CPS. Currently, there are two possible reasoning-based methodologies: the use of complete digital twins of a CPS to identify inconsistencies with reality [55] or a rigorous mathematical or logical analysis of the system [53, 56].

Further, the alerts can be forwarded to systems relating the information into attack campaigns as described in Section 6.3.

3.4 Summary

Contributions

In this chapter, we illustrated how CPS operators can identify possibilities for the deployment of anomaly and intrusion detection systems. By providing a catalog matching security requirements with appropriate monitoring and intrusion detection measures, we assist in the cumbersome choice of IDSs and their configuration. After identification, all the different IDS are joined into one coherent architecture (cf. Figure 3.1). Referring to **C1**, introduced in Chapter 1.3, we derived an architecture for aggregating needed input data and a framework combining all the methods. The combination of an SRA, derived security requirements, and finally our mapping from security requirements to monitoring and anomaly detection measures brings us towards a methodical approach for the integration of IDS in current CPS security concepts. This eases the tedious manual task of identifying suitable monitoring and detection measures as well as their correct configuration. As outlined before, the following chapters shall deepen parts of the described process and architecture. All presented monitoring or anomaly

detection systems initially require input data to work on. Hence, we present different methods for its acquisition in the following Chapter 4. Further in Chapter 5, we present a CPS-specific and protocol-agnostic high-performance approach for anomaly detection. With Chapter 6, we then develop guidance for a better understanding of the results of the deployed IDS as well as optimization strategies when going to distributed intrusion detection applications.

4 Data Acquisition

Revisiting the general idea of anomaly and intrusion detection, we need to differentiate normal and anomalous states of a system. Independent of the corresponding state representation, a scientific evaluation of any new method is only possible by application. For this, we need test input to confirm the concept versus reality. Additionally, most machine-learning methods already need training data for the actual development.

Difficulties in obtaining data

There are, however, several challenges resulting in only a small number of published datasets. First, CPS security is still a young topic with only a short research history. Hence, not too much work on data acquisition has yet been done. Second, while new projects for securing CPS flourish, these are usually under non-disclosure agreements impeding the publication of involved data. Due to a feeling of uncertainty of CPS operators, fear of unintended intellectual property release is always involved. This leads to a situation where new intrusion detection methods get developed while the data used for training or even evaluation is not public. For high-quality research, however, we must be able to re-assess previous methods to evaluate the current progress. Third, the few published datasets often lack the needed depth. As we will see in Chapter 5, accurate anomaly detection for CPS requires more than just evaluating network packet headers. However, most of the existing datasets focus on exactly this scheme. Additionally, building up testbeds of bigger CPS installations is costly.

Different kinds of data

In this chapter, we introduce two different methods for the acquisition of data.

We start with a concept allowing for capturing data in a network. At first glance, acquiring data from production systems may seem trivial. However, during projects with several industrial companies we noticed that operators still often lack knowledge of appropriate mechanisms. The main reason is usually the vast amount of data available and its transmission from complex CPS networks to an off-site analysis system. Hence, our method is designed to overcome general challenges we may face in complex CPS. The high amount of network data usually must be transferred to an off-site analysis system (also cf. Figure 3.1). Additionally, often the network link to an off-site system has a lower bandwidth compared to the CPS network. For the transport of data over

slower network links, we, thus, present a minimalist compression method keeping the additional load as small as possible.

While the use of genuine production data for the development and testing of new anomaly detection methods promises the most realistic and effective results, we cannot always rely on this method. Access to production CPS may be restricted or the publication of its associated research prohibited. Further, current standards like IEC62443 [75] call for a security-by-design approach. This encourages us to develop and integrate appropriate security measures not after the launch of a new CPS but already during its design phase. Hence, we develop another method for the generation of the required training and testing data. Using a co-simulation approach, we ensure having realistic network as well as process data. This data then reflects the actual physical process happening later in the production phase while also incorporating the effects of the underlying network infrastructure. For validation of the generated dataset, we use a simple, yet well-known anomaly detection scheme based on a fully connected neural network. While the realistic reference of the data is guaranteed by the simulation design, this way, we also assure that the dataset meets the requirements of current anomaly detection systems.

4.1 Packet-wise Compression and Forwarding of Industrial Network Captures

The previously outlined challenge can be considered a dilemma for the development of anomaly detection algorithms and machine-learning approaches. Therefore, reliable mechanisms for data probing and preprocessing are required. Parts of this chapter have previously been published in "Packet-wise compression and forwarding of industrial network captures" by Gerhard Hansch, Peter Schneider, and Sven Plaga [8].

4.1.1 Data De-duplication

In our approach for data reduction during transmission, we replace frequent occurring byte sequences using a look-up table. This look-up table can be constructed over time requiring no initial setup. Still, it allows for a decrease in network traffic over time.

Concept

For this, we identified variable and mostly static regions in TCP packets in industrial networks. For the byte sequence of the static part, we attach an `ID` to every new one encountered. Every new combination of ID and byte sequence is then transmitted once to the offsite system. We then use the ID as a key and the byte sequence as the value in

our look-up table. On later occurrences, only the ID will be sent as the corresponding byte sequence to replace is already known.

Look-up table

Internally, a prefix tree (also referred to as trie) is used to store sequences of the packets. Each node in the trie may carry an ID, which is used to replace the data sequence in a message during transmission. By using a prefix tree instead of the usual hash map for lookup-tables, the memory footprint of the table can be reduced significantly while not harming the performance. The performance advantage originates from the fact that many byte sequences only differ in few positions yielding substantial overlaps to other sequences. An example of four shortened paths in the trie is shown in Figure 4.1. The paths resemble four different network packets:

- id=1: 0x45000030
- id=2: 0x45000028
- id=3: 0x45000034
- id=4: 0x450004a8

Using a plain list for the storage, the memory footprint of these four short sequences is 16 bytes. By removing the overlap of these sequences with the trie only 8 bytes are needed to represent the same data.

Fields that are never substituted are located in the packet headers. In the IPv4 header, these fields are the total length, the identification number and the header checksum, while in the TCP header the sequence and acknowledgment number, the data offset, flags, and checksum fields are excluded. As the previously described datasets mainly consist of IP/TCP communication, other protocols were excluded from de-duplication.

In the case of encrypted data a small amount of data de-duplication is still possible, as TCP header and payload are substituted separately from the IP header. Since encryption takes place on higher network layers, the de-duplication of IP headers redeems the overhead of transmitting IDs for the TCP header and the encrypted payload over time.

4.1.2 Evaluation

Test Data and Implementation To evaluate the proposed method we implemented a demonstration based on Python3, scapy [90], and Google's protobuf [91], which we applied to the following datasets:

- simulated MODBUS traffic [41]

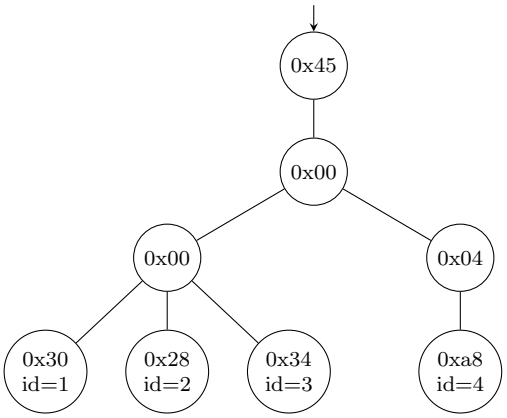

Figure 4.1: Prefix tree (trie) used for de-duplication.

- ICS/SCADA hands-on testlab [42]
- DARPA challenge datasets [40]
- Recorded data from a production robot

Our evaluation reads and iteratively processes packets from each source file. To determine the efficiency of the approach, different packet captures in the pcap format were used. Each packet is analyzed by scapy to detect the protocols and get access to the individual header fields.

According to the message format provided in Listing 4.1, the fields that are always transmitted are concatenated into a single byte sequence, which is called `static`. The remaining fields are split into two separate byte sequences. The first one contains all remaining fields of the IPv4 header and is called `ipv4_remainder`, while the second sequence, the remainder of the TCP header and its payload, is called `tcp_remainder`. These are passed to separate prefix trees, which generate a new ID for the byte sequences or return the already known one if it is present. In the case that the byte sequences are new in the tree, they are transmitted in the corresponding `_value` field shown in Listing 4.1 along with their new ID in the `_id` field. Once the sequences start repeating, only the ID is transmitted, thus omitting the redundant parts as shown in Figure 4.2.

Compression Efficiency For testing, we used our implementation to process and forward the described network captures.

Table 4.1 shows a performance comparison of data de-duplication versus the well-known Zlib compression algorithm [34]. The table indicates the original size and remaining size ratio for three different approaches, i.e., the smaller the numbers the more effective the approach.

Dataset	number of packets	original size in bytes	zlib ratio	de-duplication ratio	combined
Modbus [41]					
run8	72186	5008499	0.96	0.88	**0.86**
channel_2d_3s	383312	17449820	1.17	**0.71**	0.90
run11	72498	4955264	0.96	**0.87**	0.87
run1_3rtu_2s	305932	15870003	1.05	**0.69**	0.81
4SICS - S7 [42]					
4SICS-GeekLounge-151020	246137	18000708	0.91	**0.36**	0.46
4SICS-GeekLounge-151021	1253100	101191303	0.85	0.55	**0.53**
4SICS-GeekLounge-151022	2274747	139975880	1.02	**0.71**	0.78
Self Recorded Robot					
lab	10120	3227398	0.39	0.95	**0.36**
Office Network					
KDD'99 Day 1 [40]	1362869	280299459	0.81	0.93	**0.76**

Table 4.1: Data reduction efficiency. The most efficient approach for each dataset is highlighted by bold printing.

```
1 syntax = "proto3";
2
3 message Packet {
4   bytes  static = 5;
5   uint32 ipv4remainder_id = 9;
6   bytes  ipv4remainder_value = 1;
7   uint32 tcpremainder_id = 3;
8   bytes  tcpremainder_value = 2;
9 }
```

Listing 4.1: Protobuf specification of the message format.

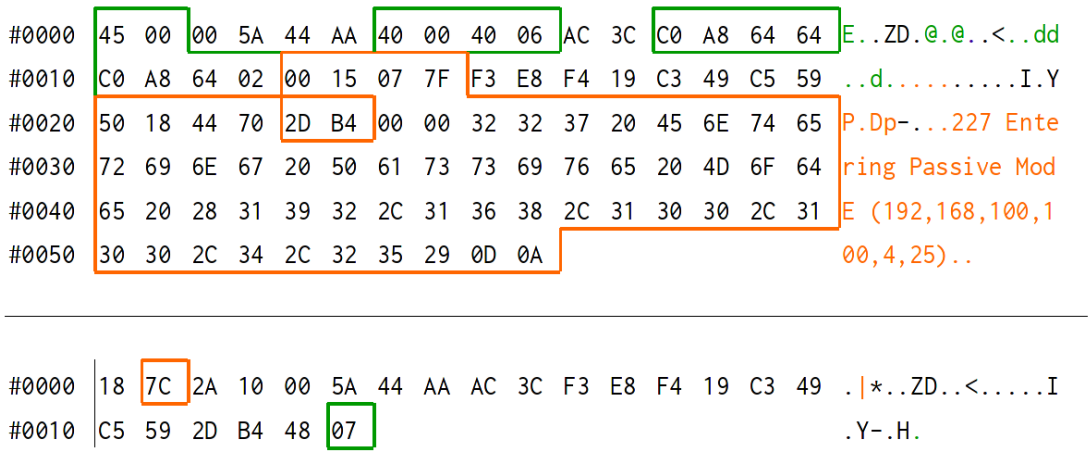

Figure 4.2: Example of the de-duplication effect on an FTP packet captured in an industrial context. The upper part resembles the original packet while the bottom part displays the de-duplicated packet. The colors highlight the substituted parts of the packets.

The first approach, called *zlib*, uses packet-wise Zlib compression. As can be seen from the indicated ratios for the different datasets, the efficiency is varying severely and can even lead to an increase of data in some cases. The self-recorded dataset from a robot running in a laboratory yielded a very low compressed size of only 39% of its original size. As Zlib compresses bytes inside a fixed search window, local redundancies can be removed more efficiently. However, for the remaining datasets the packet-wise Zlib compression reduced the data size only in a few cases while in some the resulting packets even increased. This is caused by the fact that especially the industrial network packets like Modbus and S7 communication tend to be quite short. For these short packets, an effective compression using stateless Zlib is simply not possible.

The results of the proposed de-duplication method using a lookup-table is shown in the next two columns of Table 4.1. Except for two cases, our method achieves smaller packets and therefore a higher compression ratio than the previously described Zlib approach. It is worth noting that although our method still involves some overhead and theoretically can lead to an increase in the data volume, in practice, none of our test datasets shows such an increase.

Finally, chaining both approaches was tested by first de-duplicating the data as described foremost and then using the Zlib compression on top of the remaining packets. As can be seen by the results, this combined approach is always more efficient than just using plain Zlib alone. However, in many cases using Zlib on top of the de-duplicated

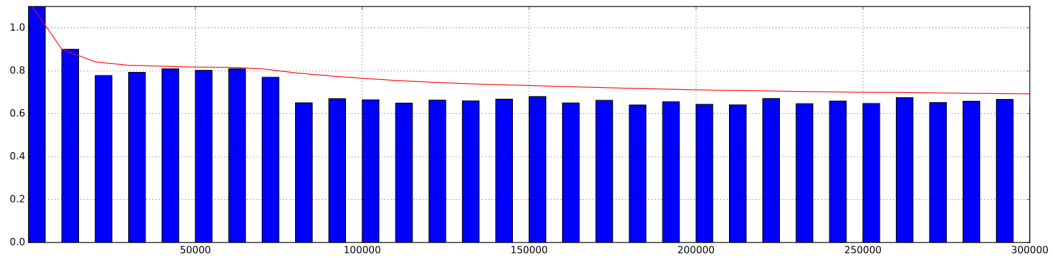

Figure 4.3: Compression ratio over packet count. The dataset contains generated modbus traffic with 3 remote terminal units by [41].

packets even increases the size again. It is assumed that this is caused by the fact that for very small packets the Zlib compression does not work as good as expected.

Whereas Table 4.1 shows the average compression ratio for each dataset, we also studied the time needed for our method until efficient compression starts. In the beginning, the compression method includes an overhead. As on every first occurrence of a new byte sequence an ID is added to the packet, the overall transmitted data is slightly increased. Figure 4.3 shows for every batch of 10000 packets the actual compressed size of these packets in one of the captures in the Modbus dataset. After just 10000 network packets we already see a decrease of the overall data to be transmitted. This indicates that the previous assumption about frequent repetitions in industrial network packets holds up. It takes only about 80000 network packets to settle at a compression level of about 65%. Looking into the respective actual network capture, this corresponds to about 932 seconds ($\approx$ 15 minutes) of network traffic in this setup. After this, the compression level of further batches of packets stays almost constant.

4.2 Realistic Data Generation for Anomaly Detection in Industrial Settings using Simulations

While the previous approach of data compression is useful once CPSs have been deployed, it cannot be used to construct datasets of yet undeployed systems. As described earlier, industrial standards like IEC62443 [75] emphasize the early integration of security measures. As CPSs are highly specific to their use case, we cannot always rely on training anomaly and intrusion detection systems on data of other systems. Therefore, we present a more general approach for deriving suitable datasets in this chapter. By basing our method on system models which may be available in early development stages and also before actual deployment of a CPS, we enable training of detection systems already adapted to the later use case. In addition, this approach for generating reproducible and shareable datasets is also beneficial to anomaly detection research in general.

Parts of this chapter have been published in "Realistic Data Generation for Anomaly Detection in Industrial Settings using Simulations" by Peter Schneider and Alexander Giehl [9].

Anomaly detection as a solution

Therefore, relying on signatures of the attack vectors seems inappropriate. As these fingerprinting-based techniques are inferior in this scenario, machine-learning based anomaly detection is a viable solution for security in industrial settings. These systems rely on a model of normal system operation. Attacks are assumed to change the system behavior and can, therefore, be detected.

Lack of data

One major problem in developing anomaly detection systems for this setting is the lack of suitable training data. Nearly all existing approaches do need a lot of data to derive a model of normal operation or even distinguish between normal behavior and anomalies. However, there is only a limited set of data available since almost all companies having such data fear to expose their intellectual property with it—maybe even unknowingly. Hence, most of the current research uses either private, handcrafted, or inadequate datasets to evaluate their approach [43]. Additionally, only a few of them are publicly available. This results in state-of-the-art intrusion detection systems which cannot be compared by their performance. Therefore, there is a need for suitable training data for detecting anomalies in industrial networks and enabling big data approaches for IT security in this domain.

Data generation using simulations

In this section, we present an approach to develop meaningful but reproducible datasets which are suitable for process- and network-based anomaly detection. By the use of process-based simulations, special network data is crafted that can represent

data in real-world CPS. Our approach is not only capable of generating cyber-physical process data but also of simulating attacks on the underlying processes. Having control over the simulation configuration, all the data is labeled in the end and is, therefore, usable for state-of-the-art anomaly detection systems.

In summary, we make the following contributions:

- We present a workflow to generate cyber-physical process data for anomaly detection systems on a large scale.
- We evaluate that data and investigate its usability for the detection of anomalies.
- We show how to include attacks in the simulation and how to detect them in the data using state-of-the-art methods.

4.2.1 Simulation Framework

Framework concept

The simulation framework proposed consists of three elements. At first, a process simulation generates sequences of process parameters based on realistic system models. The foundation for this process simulation is a process model describing the available components and their interaction. The simulation calculates the different process parameters based on mathematical and physical models. Thereby, the process parameters are sampled at specific time intervals. Afterwards, the physical process model is split into parts and mapped to virtual devices resembling the network components monitored. As process simulations often also use object-oriented programming paradigms, their definitions also use inheritance and abstractions. For our framework, we assume that every single object in the process model represents a single networked device in the real world. The abstractions introduced by inheritance in the process model result in a reasonable amount of devices. At the same time, the capsulation of part of the process in classes hides some of the internal process parameters, which in reality are not communicated over the network.

Network Architecture

Furthermore, without loss of generality, we assume that those devices are connected in a bus topology, as this mode of operation is currently being adopted in industry connectivity [66]. The extracted components become the networking nodes in the network simulation. There, we use one application for each network node sending and receiving the process data between these components. The generated process parameters are embedded in a suitable Fieldbus protocol. In fact, for the Fieldbus protocol, it is possible to choose one according to the needs of the dataset to be generated. Figure 4.4 shows the general workflow for this sequential simulation framework. The *process model* together with its *input parameters* are used in the *process simulation framework* to de-

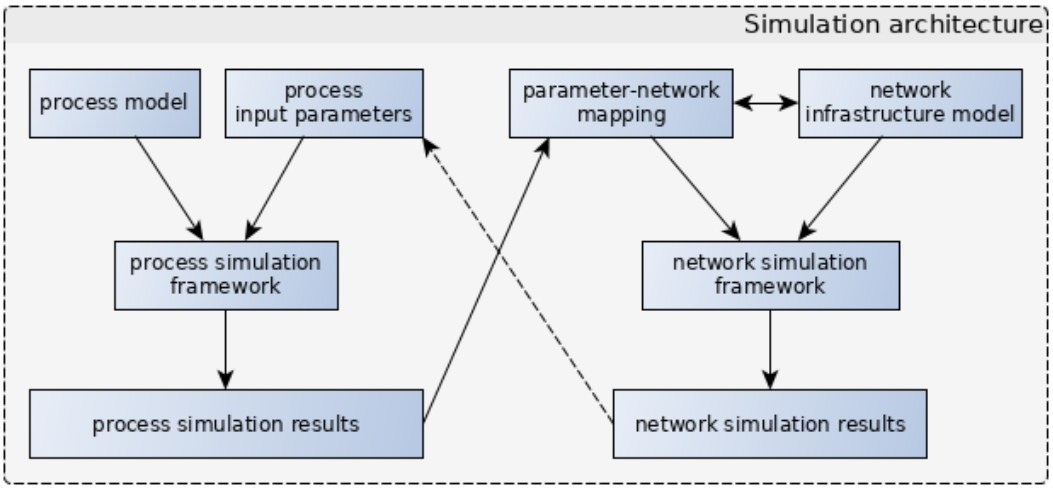

Figure 4.4: Simulation framework workflow.

rive process values as a *result*. This result is a list of several process values for every time step. Every such value is then assigned using a *parameter-network mapping* to its network node. The available network nodes are defined in the *network infrastructure model*. After this, the *network simulation framework* can produce network traffic corresponding to the simulated networked process.

If the effects of the networking onto the process itself, e.g., latency or jamming, need to be simulated, a co-simulation is required instead of the sequential approach outlined before. In a co-simulation, as shown in Figure 4.4, communication between the two simulation frameworks also uses the dashed line. Here, the simulation stops after each time interval to include the feedback of each other simulation. Both simulations are executed alternating while an adapter transfers the states between the simulations [92]. In the end, this yields realistic network data very similar to captures in a production site with real data.

So, in summary, an operator of a CPS can derive a realistic dataset by following these steps:

- Obtain or create a model of the physical process in the modelica language (see Section 4.2.1.1).
- Apply the automatic infrastructure derivation to generate a network mapping (see Section 4.2.1.4).
- Define a prototype application in NS-3 for the fieldbus to use and execute the network simulation (see Section 4.2.1.2).

If desired, the steps can be repeated with a modified version of the initial process model after applying model modifications as introduced in Section 4.2.1.5.

4.2.1.1 Industrial Process Simulation using Modelica

Modelica is a popular object-oriented programming language used in a variety of process simulation frameworks. It is designed to model physical processes and control loops easily. OpenModelica [93] is an open source implementation of that language providing a graphical modeling toolkit as well as interfaces for interaction with the simulations.

Building blocks

Physical processes are modeled in the Modelica language using a variety of building blocks. Unavailable functionality can be added through the addition of external libraries.

Relation to real-world systems

The modeled process is then compiled to an executable that generates corresponding time-series data to given initial values. The data generated by OpenModelica using free models has already been reviewed [94, 95] and found to be useful for analysis of real-world systems. Building on these findings, OpenModelica provides the industrial process simulation to the framework.

Introduction to Modelica

As Modelica is an object-oriented language, the definition of this simulation starts with its parent element followed by the constituent components. Each of the listed components corresponds to a separate Modelica language model definition. They resemble building blocks as the physical and mathematical model is encapsulated in them. Due to this encapsulation, Modelica uses a layered system description. The blocks shown in Listing 4.2 (l. 3–6) constitute the top-most layer of this system description. To model the system behavior, the components are connected through `equations`. In the top-most simulation description, like the one shown in Listing 4.2, the only equations are connections from output to input values (l. 9–13).

```
1  model ClosedLoopSimulator "Plant simulation [..]"
2    extends Modelica.Icons.Example;
3      Models.HRBPlant Plant([..]) annotation([..]);
4      Modelica.Blocks.Sources.Step ValveOpening([..]);
5      Modelica.Blocks.Continuous.PI TempController([..]) [..];
6      Modelica.Blocks.Math.Feedback Feedback1 [..];
7      Modelica.Blocks.Sources.Step TWOutSetPoint([..]) [..];
8      inner System system annotation([..]);
9    equation
10     connect(ValveOpening.y, Plant.ValveOpening [..]);
11     connect(TempController.y, Plant.GasFlowRate) [..];
12     connect(Feedback1.y, TempController.u) [..];
```

```
13      connect(Plant.WaterOut_T, Feedback1.u2) [..];
14      connect(TWOutSetPoint.y, Feedback1.u1) [..];
15  end ClosedLoopSimulator;
```

Listing 4.2: Excerpt of an HRB plant model written in the Modelica language [95]. The regions indicated by [..] have been shortened.

Additionally, OpenModelica provides means for interfacing with the simulation also allowing the implementation of a co-simulation framework.

4.2.1.2 Network Simulation using NS-3

For the purpose of simulating the underlying network infrastructure, NS-3's python interface is used [96]. It provides the necessary simulation of the physical communication channels as well as the lowest network layers up to TCP. The industrial network data is generated using a custom application layer protocol that is common in proprietary industrial settings. Using this protocol the application simply sends messages containing the parameter name and value to the destination component at regular intervals. That is the same methodology as in common industrial Fieldbus protocols. While CAN uses CAN-identifiers transferred along with the actual data values, the high-level CANopen protocol uses a known dictionary, the CANopen Object Dictionary (COD), to map bytes into the transferred Process Data Objects (PDOs) [85]. In the end, both methods define a static mean to look up the meaning and interpretation of every data byte transmitted, effectively yielding name-value pairs. The same is true for the widespread Modbus protocol, which uses function codes transmitted along with the actual data [97].

4.2.1.3 Sequential Simulation

To derive a multilayered dataset with realistic process data on the one hand and a realistic embedding in industrial network traffic, on the other hand, we need to combine these two simulation frameworks as shown in Figure 4.4.

Using this combined approach we assume that prepared anomaly detection algorithms will be able to generate alarms more accurately and to detect attacks originating from network manipulation or the manipulation of involved processes.

For this, the cyber-physical process under test needs to be modeled in the Modelica programming language. As such models often are part of the engineering and development process, we assume that such a model is detailed enough to allow analyzing the main functionality. For research, there are also several thoroughly tested open source

models available which can be used for comparison of IDS performances [94, 95, 98]. Each of these models requires a stable initial state. This state is not required to be an equilibrium state for the industrial process, but it should not be a state which causes the system to diverge. Using the model and its initial state, it is possible to run a simulation with OpenModelica resulting in all process parameters at given time intervals.

4.2.1.4 Automatic Infrastructure Derivation

Deriving network data from the generated process parameters requires a network infrastructure model corresponding to the cyber-physical model. The framework relies on the assumption that every building block in the Modelica description of the industrial process at the top-most level represents a single network node, i.e., a discrete device. This allows for more realistic modeling of real-world scenarios where parts of the functionality of the system are encapsulated within one single complex device. Not all simulated connections from the OpenModelica simulation will show up in the final network trace as in real-world scenarios where the system is only partially visible to an IDS. As the Modelica model also describes the interconnection of blocks, the accompanying data exchange between these follows directly from the model. A typical block to block connection is expressed as shown in Listing 4.3.

```
1 connect(TWOutSetPoint.y, Feedback1.u1) annotation(
2     Line(points = {{-69, 10}, {-46, 10}},
3          color = {0, 0, 127}))
```

Listing 4.3: Modelica description of a block to block connection.

Algorithm 1 automatically splits the Modelica model at its topmost level into blocks and returns a list of the connections between these blocks. With the use of regular expressions, it is possible to parse the model description without implementing an interpreter for the Modelica language. The expressions `\s*model m` (l. 5–7) and the `\s*end m` (l. 13–15) find the start and corresponding end of the description of model m. Using two capture groups every pair of two connected components and also the direction of the connection can be retrieved with `connect\(([^,]*), ([^)]*)\)` (l. 9).

The referenced function `match` takes a pattern and a string that is searched for the pattern. Concerning the regular expressions, the `*`-modifier is assumed to be greedy

Algorithm 1: ModelSplitting

Input: Modelica model file M
Input: model name m
Output: List of top-level connections L

```
1  Let L be a list;
2  startFound = endFound = False;
3  foreach line l in M do
4  |  pattern = "\s*model " + m;
5  |  if match(pattern, l) then
6  |  |  startFound = True;
7  |  end
8  |  if startFound ∧ ¬endFound then
9  |  |  pattern = "connect\(([^,]*), ([^)]*)\)";
10 |  |  res = match(pattern, l);
11 |  |  if res ≠ None then
12 |  |  |  append(L, group(res, 1), group(res, 2));
13 |  |  end
14 |  end
15 |  pattern = "\s*end " + m;
16 |  if match(pattern,l) then
17 |  |  endFound = True
18 |  end
19 end
20 return L;
```

and the function `group(res, i)` returns the content of the i-th capture group in the matching result `res` (l. 10–12).

The execution of this algorithm on the model given in Listing 4.2 with the model name `ClosedLoopSimulator` returns the list of connected component pairs as

$$\begin{aligned} L = \{&(ValveOpening.y, Plant.ValveOpening),\\ &(TempController.y, Plant.GasFlowRate),\\ &(Feedback1.y, TempController.u),\\ &(Plant.WaterOut\_T, Feedback1.u2),\\ &(TWOutSetPoint.y, Feedback1.u1)\} \end{aligned}$$

Based on the block names of each connected component, i.e., the part up to the dot, there are five nodes for the network simulation: `ValveOpening`, `Plant`, `Feedback1`, `TempController`, and `TWOutSetPoint`.

Data exchange between the network nodes can be modeled either by push or pull paradigms. From our experience, both methods are actually used so that we decided to push the data to the next node as most field buses (e.g., CAN or ProfiNet) are typically used like this. A mapping component splits the simulation results into the desired parts, which generates a list of data packets, their destinations, and timestamps for each identified node. These lists are then passed to a custom application for the NS-3 network simulation, which sends out the corresponding network packets at the right time. Being an NS-3 application, the simulated network stack handles the underlying protocols, i.e., TCP handshakes and responses, while on the application layer the data is not used any further. By instructing NS-3 to capture the network traffic on each virtual device into packet capture files, simulated network traffic traces are generated. Based on the integration with the process simulation and the custom application, this data is backed with realistic data from an industrial process as it might have been recorded in an actual facility.

4.2.1.5 Sensor Manipulation Attacks

For the creation of suitable test data for anomaly detection systems, it is indispensable to also have malicious examples, i.e., data corresponding to the attacked system state. Therefore, the framework provides a generic algorithm to manipulate the simulation model. Algorithm 2 replaces one connection in the model with a time-triggered switch to model a manipulated sensor or actuator. The given connection between the output value c_1 and input value c_2 gets replaced by a connection from c_1 to the first input of the switch s_1 and the output of that switch s_y to c_2 (l. 11-12). The value n used for replacement during the manipulation is connected to the second input of the switch s_2 (l. 13). Using an integer step function is, the inputs get switched at a specific point in time t_2 (l. 14).

$$is = \begin{cases} 1, & for \quad t \leq t_2 \\ 2, & for \quad t > t_2 \end{cases}.$$

Therefore, the input value of c_2 equals to

Algorithm 2: Automatic Modelica model manipulation for inclusion of attacks.

```
   Input: Modelica model file M
   Input: model name m
   Input: attacked connection c1, c2
   Input: manipulation node n
 1 startFound = endFound = False;
 2 foreach line l in M do
 3     if match("\s*model " + m, l) then
 4         startFound = True;
 5     end
 6     if startFound ∧ ¬endFound then
 7         if match("\s* equation", l) then
 8             insertBlock(integerStep);
 9             insertBlock(extractor);
10             print(l);
11             connect(c1.y, extractor.u[1]);
12             connect(extractor.y, c2.i);
13             connect(n.y, extractor.u[2]);
14             connect(integerStep.y, extractor.i);
15         else
16             res = match(".*connect\(c1, c2\).*", l);
17             if res == None then
18                 print(l);
19             end
20         end
21     end
22     if match("\s*end " + m,l) then
23         endFound = True;
24     end
25 end
```

$$c_2 = \begin{cases} c_1, & for \quad t \leq t_2 \\ n, & for \quad t > t_2 \end{cases}.$$

Applying this algorithm to an existing Modelica model yields a similar model representing an attacked version of the system starting from the time t_2. An example of the application is shown in Section 4.2.2.1.

4.2.2 Data Usability Validation

The usability of the generated data from such simulations for the purpose of anomaly detection was analyzed by generating a test dataset and training a deep learning network with it. The systems analyzed for validation are a heat recovery boiler (HRB) plant [95] from the *ThermoPower* library (with 1367 simulated equations), a velocity control system for a drive [98] from the *IndustrialControlSystems* library (258 equa-

tions) and a complex wastewater treatment system [94] from the *WasteWater* library (3066 equations). In the following, a more in-depth look is given on the HRB plant, while similar observations and results can be obtained for all tested models.
Figure 4.5 shows the *ClosedLoopSimulator* for the HRB plant. The system consists of an HRB plant (P), a temperature controller (*TempController*) and the input values for the setpoint temperature, i.e., the temperature the water leaving the plant shall have, and the valve opening, i.e., a parameter steering how fast the water is running through the boiler.

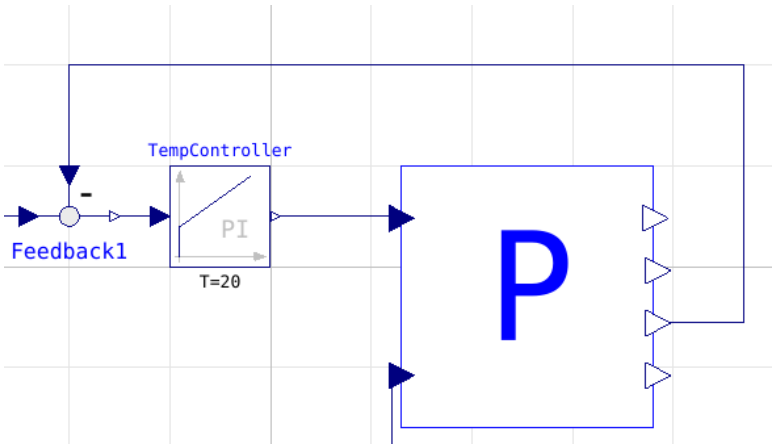

Figure 4.5: Original system of a HRB plant in Modelica. [95].

The building block of the plant encapsulates the whole process of incoming gas and water to the boiler, the heat transfer between these two, and the water flow rate. However, the actual control of this process, i.e., the regulation of the heating based on the desired water flow rate and water temperature, is placed outside the plant. This process is steered by the mentioned inputs and the sensor values.

This process is comparable to many industrial applications where a user controls a machine via its interface over the network. Our simulation of the system from [95] using OpenModelica yields the temperature curve for the outgoing water shown in Figure 4.6. The blue curve shows the desired temperature (in Kelvin), i.e., the input signal over the network, while the red one shows how the simulated boiler control system reacts. At the time $t_1 = 50s$, the valve is closed a bit to test the response of the controller to disturbances in the process. The water temperature gets back to the setpoint after a short time.

Then, at time $t_2 = 200s$, the setpoint value for the water temperature is changed causing the HRB plant to heat the water up. This leads to the increasing water temperature (red curve) later settling at the desired temperature.

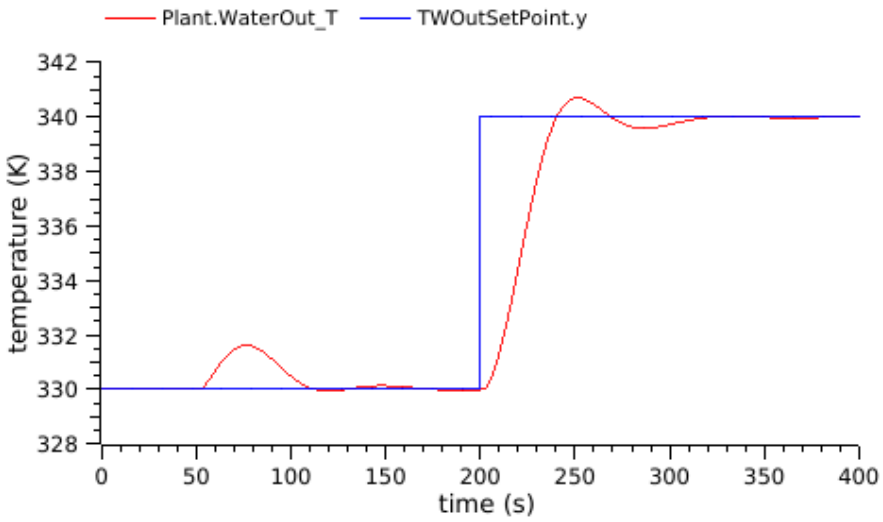

Figure 4.6: Temperature curve of original simulation.

4.2.2.1 Introducing simulated attacks

To show the feasibility of the integration of attacks into our simulation framework, we implement a manipulated sensor attack. For this, we altered the initial model to simulate a manipulated sensor device firmware. Starting from the time $t_2 = 205s$, we exchange the real output temperature of the plant with the actual temperature setpoint. This directly resembles an attack on the temperature sensor of the boiler. For example, a specially crafted malware which forces the sensor to report wrong values might trigger this scenario. This attack can be modeled with the Modelica language by exchanging the input to the feedback unit $f1.u1$ from the water temperature $P.WaterOut_T$ to the actual setpoint temperature $TWOutSetPoint$. The application of the generic Algorithm 2 yields the modified part of the model shown in Figure 4.7. A switch ($extractor1$) is used to choose one of the two incoming signals. The real sensor value is assigned to the first input, the setpoint temperature to the second. The choice of the output is derived from an integer step function yielding the input to the feedback unit as

$$f1.u1 = \begin{cases} P.WaterOut_T, & for \quad t \leq t_2 = 205s \\ TWOutSetPoint, & for \quad t > t_2 = 205s \end{cases}.$$

The simulated temperature curve of this modified model is depicted in Figure 4.8. As expected, the water temperature now settles at $333K$ instead of the desired setpoint of $340K$. Starting from $t_2 = 205s$, the closed-loop controller gets the signal that the desired temperature is already reached and therefore stops further heating. In this case,

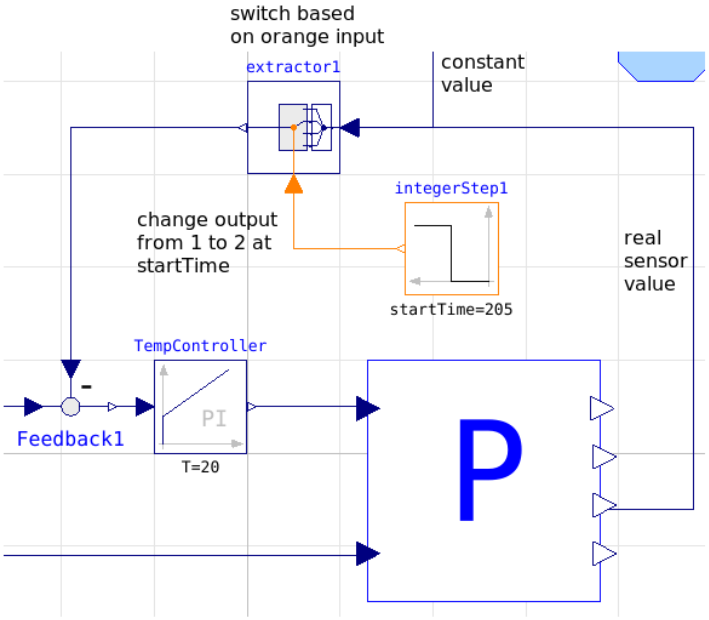

Figure 4.7: Faked sensor data model.

this leads to an early stop of the heating. If the sensor incorrectly reports a temperature below the setpoint, this results in a diverging system where the temperature controller never stopped heating.

4.2.2.2 Anomaly Detection on Simulated Data

For the development of models based on machine-learning, there are requirements to generated datasets to be met. At first, the amount of available data to train a model hugely impacts the possible results. Then, there is a need for diversity in the data as using redundant examples cannot improve a model's reliability [99]. To test whether the generated data is not just plausible but also suitable for machine-learning, we used a neural network to predict the next parameter values given the current ones.

In general, the workflow for training and evaluating anomaly detection systems on the generated data is shown in Figure 4.9. At first, normal operation data is generated using the framework explained in Section 4.2.1.3. In the next step, it is required to extract meaningful features from the generated data. In the case of analyzing the network traffic, this includes parsing the protocols and identifying suitable data fields for the detection approach. Thereafter, one generated dataset is used to train the anomaly detection approach. The approach's performance can then be evaluated with a second generated dataset.

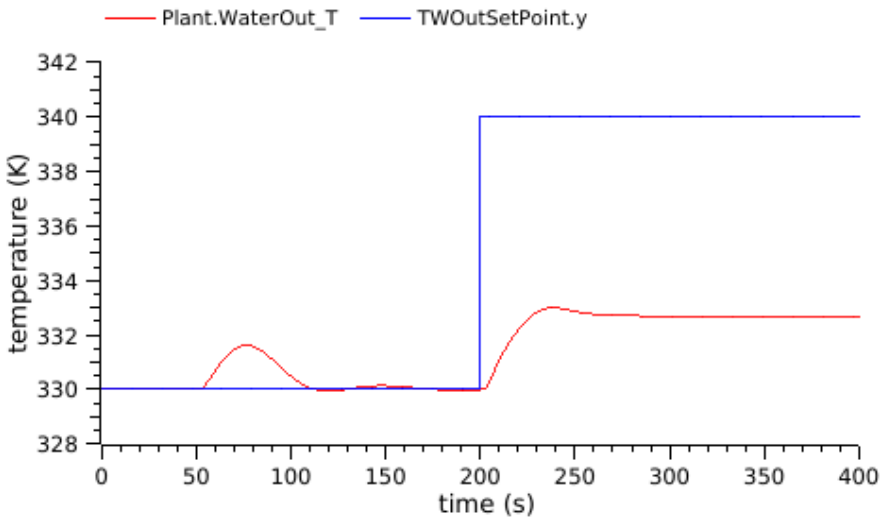

Figure 4.8: Temperature curve of attack simulation.

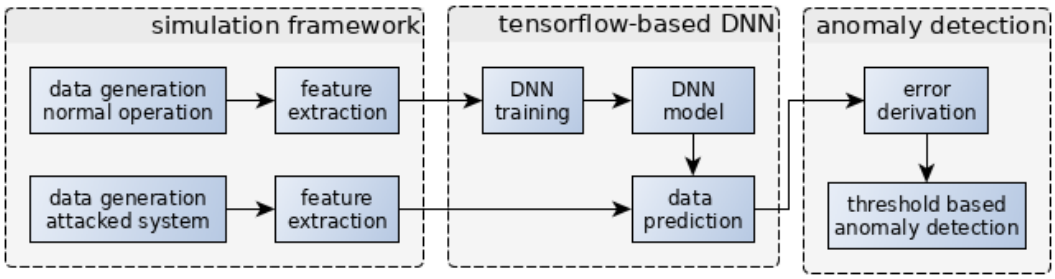

Figure 4.9: Workflow for anomaly detection using simulated data.

From the 1169 process parameters available after the simulation of the HRB plant in OpenModelica, we only use a small subset that may be directly available in network traffic, i.e., the input and output connections of the plant and the two controllable inputs, the setpoint temperature and the valve opening. These parameters are listed in Table 4.2. To automate this, we split the Modelica model using Algorithm 1 at its topmost level into building blocks, i.e., the blocks shown in Figure 4.5. The connections between these blocks are interpreted as network communication.

In fact, this is the same split as for the mapping of process data to network devices described in Section 4.2.1.4. As we do not include network metadata in this analysis, the wrapping of the data into network traffic only leads to a different representation of the same data. That is the reason it can be omitted here.

Parameter	Meaning
`ValveOpening.y`	User-defined opening of the plant's valve
`TempController.y`	Output value of the temperature controller steering the gas flow rate of the plant
`Plant.WaterOut_T`	The sensor reading for the water temperature leaving the boiler
`TWOutSetPoint.y`	User-defined temperature of the water

Table 4.2: Derived parameters and their meaning.

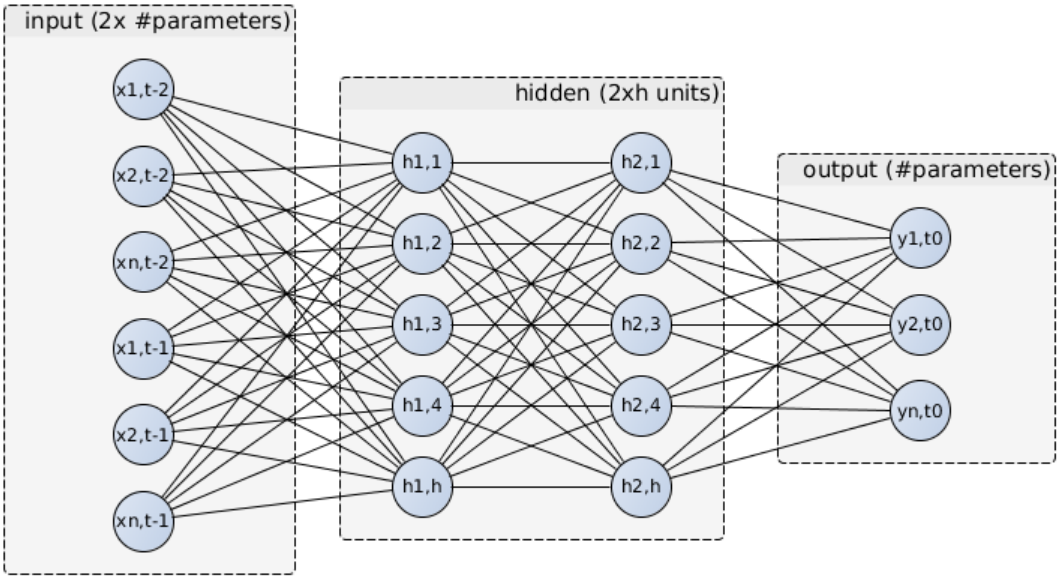

Figure 4.10: Neural network architecture.

The anomaly detection approach for validation is based on a Deep Neural Network (DNN) architecture shown in Figure 4.10. The network is used to predict the values of the process parameters at one time step ahead. By observing the difference between the real and predicted values, the state of the industrial process can then be monitored. The DNN uses linear regression layers to predict the values of the process parameters in the future.

A normal regression layer uses the formula

$$y = X \cdot W + b,$$

whereas X is the input vector, i.e., the current process parameters values, W a weight matrix and b some constant vector. As the process is designed to lead to a stable

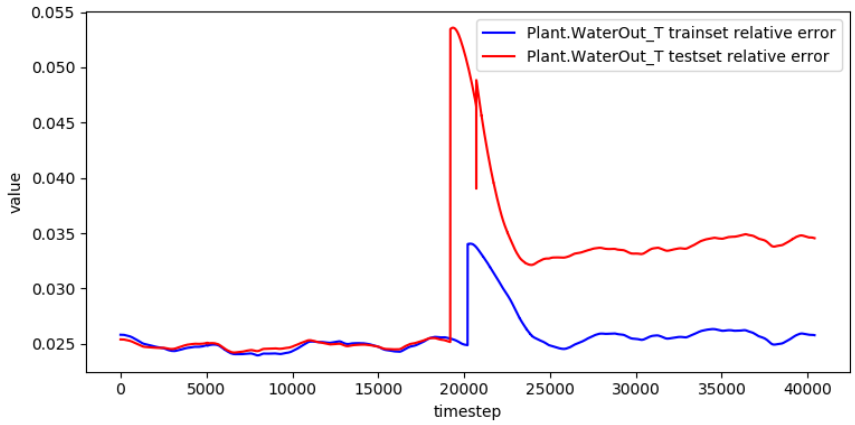

Figure 4.11: Relative prediction errors for the normal and attacked dataset of the water outflow temperature.

system, often the input and output values remain almost constant. To ensure the model does not rely on these values being constant, an additional constraint was added to the model with $b = 0$ for all times. Apart from this modification, the neural network is constructed as usual for multi-dimensional regression tasks.

Our neural network consists of one input, one output, and two hidden layers. The input layer has the size of two times the number of parameters being predicted in the output layer. The size of the hidden layers is adapted to each evaluated model as the more complex models require predicting more parameters and therefore require the network to cover more information. In addition to the original definition of the system model, some normal distributed noise was introduced on the valve opening input. This leads to a continuous fluctuation in the water outflow of the plant simulating imperfections in the implementation which were missing in the original system model.

The variations are introduced in the data this way to facilitate the use of machine learning algorithms. Without these variations, effective learning is not possible as misleading assumptions could be derived due to overfitting. With the data from the unmodified operating HRB plant, the network is trained to predict the values of the input connections at the next time step, t_1, based on the previous, t_{-1}, and the current, t_0, values. So, the network is learning the normal system behavior without the knowledge of how an attack might look like.

After training using the gradient descent method, a comparison of the predicted and real behavior results in the blue relative error curve for the out-flowing water temperature in Figure 4.11. The error of the training dataset remains in a constrained band. The only exception is the short peak starting at time step $20000 (= 200s)$. This

setting	training data	test data
initial setpoint temperature	$330K$	$320K$
t_1, time for setpoint change	$200s$	$190s$
t_2, time for attack	$-$	$205s$
final setpoint temperature	$340K$	$350K$

Table 4.3: Simulation setup for the generation of datasets.

is when in the unmodified version of the model the setpoint temperature is altered. Therefore, it is correct to interpret this spike as an anomaly. The interpretation, whether this anomaly is actually related to an attack, is outside of the scope of this data usability validation.

4.2.2.3 Attack Detection using Simulated Data

We also investigated whether simulated attacks can actually be detected while the model has been trained only with normal data.

Therefore in a second run, we simulated the manipulated sensor model depicted in Figure 4.7. Additionally, we changed the initial setpoint temperature, the time when the setpoint change occurs, and the amount of change. Therefore, if the model was overfitted, the relative error should increase already before the manipulation starts because the initial and final setpoint temperatures are different. The configurations of the test and training simulations are shown in Table 4.3.

The previously trained model was then used to predict the parameters' values at each next timestep. The red curve in Figure 4.11 illustrates the relative difference in predicted and real values. For the first part, the error is of the same order as for the unattacked (blue) dataset. This verifies that the learned model is actually portable to similar situations as the attack is only carried out after $t_2 = 205s$, i.e., time step 20500. The spike at time step $19000 (= 190s)$ corresponds to the change of the setpoint temperature. As expected, the peak on the test dataset occurs earlier than in the training dataset. This indicates that a configuration change of the plant does not alter the prediction capability of the learned model.

While with the unattacked dataset the model comes back to an error near to zero, with the attacked dataset the relative error levels at an about ten times higher relative error caused by the manipulated sensor. From the time $t_2 = 205s$, the water outflow temperature sensor starts maliciously reporting that it measures the setpoint temperature. Thus, the heating stops too early and the outflow temperature has a significant

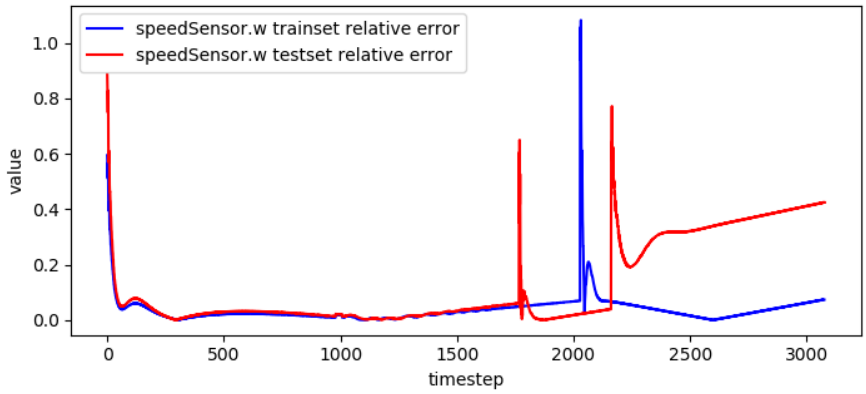

Figure 4.12: Relative prediction errors for the normal and attacked dataset of the speed sensor in the *IndustrialControlSystems* VelocityDrive simulation.

difference to its actual setpoint. This difference can now also be seen in the right half of the red curve in Figure 4.11.

To derive a suitable anomaly detection approach, an error threshold can be estimated either by experimental evaluation or by appropriate machine learning strategies. This is possible since, given the simulation setup, each time step in the data is labeled.

Similar results can also be obtained for other models. Figures 4.12-4.13 show the relative error curve of one of the parameters in the *VelocityDrive* and *WasteWater* simulations. The simulation configuration for the training and test datasets has been altered in a similar way as shown in Table 4.3 to verify the model portability.

During about the first half of the test datasets, only already available constraints have been altered to test model portability while towards the end of the simulation an attack is introduced using the Algorithm 2. As in the HRB example explained before, the blue curve always represents the relative error on the training data, i.e., with no manipulation, while the red curve shows the error of the test data introduced by model configuration changes and attacks. Also in these two more complex simulations, a sensor manipulation can be detected while the neural network has only been trained on normal operation data.

For all the models tested, a suitable size for the hidden layers in the DNN was chosen by experiments. As the focus of this analysis lies on the validation of the usability of data generated by process simulations, these values are not fine-tuned for best performance.

Table 4.4 lists the number of simulated equations per model, the parameters extracted for prediction in the DNN, and the size of the hidden layers used.

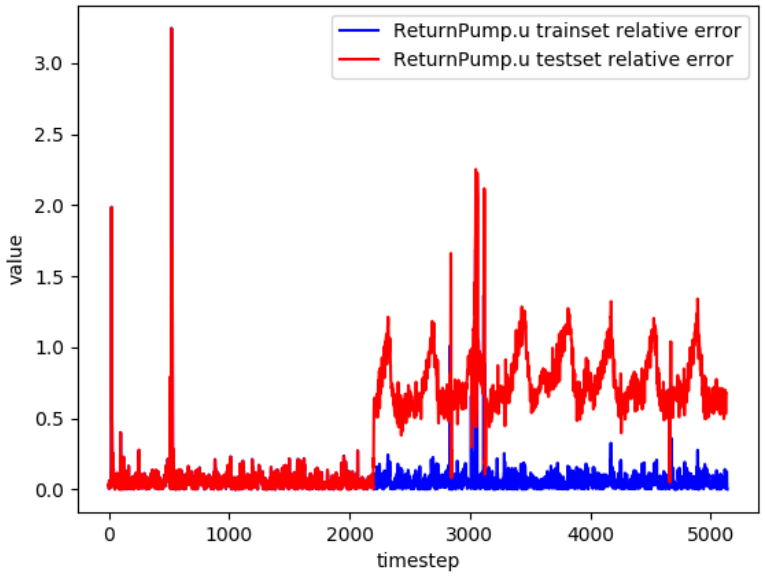

Figure 4.13: Relative prediction errors for the normal and attacked dataset of a pump inflow in the *WasteWater* ComplexPlant simulation.

model	simulated equations	parameters for DNN	size of hidden layers
HRB plant	1367	5	5
VelocityDrive	258	20	17
WasteWater	3066	21	15

Table 4.4: Simulation complexity for the validated models.

4.2.3 Conclusion

In this chapter, we presented a framework and workflow to generate usable industrial anomaly detection data. By using a combination of modeling, simulation, and an infrastructure mapping we can create industrial network traffic that reflects the physics of the network transfer as well as those of the cyber-physical process. Being a simulation, our approach does not require costly specialized hardware.

Additionally, we showed that the integration of attacks in the simulation results in labeled data suitable for machine learning. In contrast to solutions that use hardware or hardware-in-the-loop, we can model an arbitrary complex system while still having a scalable system. Also, simulation of attacks on the cyber-physical system can be

carried out without interference with production environments or danger of physical impact.

In our evaluation, we additionally showed that it is actually possible to train anomaly detection systems to predict cyber-physical systems behavior. Given that, new possibilities emerge by the integration of this knowledge into existing intrusion detection systems.

Previous approaches designed the anomaly detection to be suited for the use case, i.e., the approach has been adapted to the domain's constraints and specific problems. Instead of that, we used off-the-shelf approaches to detect anomalies in the data. Therefore, for the anomaly detection itself, there is no need for a deep understanding of the underlying cyber-physical process.

4.3 Summary

Contributions

In this chapter, we detailed two methods for the acquisition of industrial network data. Thereby, we focus on achieving realistic and meaningful datasets. As the best option to approach always is to use data from real-world CPS installations directly, we introduced a compression method in Chapter 4.1 facilitating the handling of large amounts of that data.

Real-world projects, however, often require the CPS operators to develop security measures early during development phases where actual CPS may still be lacking. For this situation, we present a framework of building-blocks to construct a co-simulation of the CPS under development. The incorporation of physical models, a possible IT infrastructure, and the corresponding process and IT network simulations yields a comprehensive solution for the creation of realistic dataset generation. Additionally, we also show how attacks can be introduced allowing for a detailed study of the process data during an ongoing attack. As we base our systems on simulation, we can carry out even advanced attacks without putting real systems to risk.

Recalling the promised contribution **C2** in Chapter 1.3, these two methods and the identified datasets shall provide a common basis for future analyses on new methods for anomaly detection in cyber-physical systems. By the use of our proposed simulation framework, we also allow for studying new attack schemes that have not yet been captured in real installations. Further, formulating a framework for the creation of the dataset enables others to reproduce the same dataset or expand on them to advance the whole research field.

Relation to next chapter

With a common data basis at our hand, we now continue the journey of IDS deploy-

ment in CPS by investigating suitable anomaly detection methods. In the following Chapter 5, we, thus, present a new anomaly detection method focusing on extending the two points already supported by our data acquisition methods. First, we focus on a high-performance approach to utilize the generated benefit of the packet-wise compression scheme. Second, we switch our interest to network packet payloads instead of their metadata to emphasize the relation between physical and real-world data.

5 Model-based Anomaly Detection

Network- vs. host-based detection

As presented in Chapter 2, there are different types of anomaly detection currently known. Mainly researchers distinguish between network-based and host-based detection methods. Their names refer to the main data source used for the detection method. For host-based anomaly detection, the state of a system is usually derived from data recorded at one specific IT system. Instead, network-based methods utilize data captured on associated network links. Since the internal design of CPS is usually non-public, little information on augmenting or orchestrating their operating systems is available. Further, expensive certification requirements limit the possibilities of adding information extraction capabilities to the hosts themselves. Therefore, we see the larger impact for network-based anomaly detection in industrial use cases. Also, network monitoring is often already in place allowing for easy extraction of required data.

Signatures

Another widespread paradigm in anomaly detection is the use of signatures of known malicious code fragments or network packets. Changes from one attack wave to the next and so far unseen attack codes make it effectively impossible to create such signatures. For the current situation, adapted anomaly detection methods, hence, should not rely too much on the signatures. In the case of network-based anomaly detection, this means we cannot rely on pure whitelisting approaches.

Parts of this chapter have already been published in "High-Performance Unsupervised Anomaly Detection for Cyber-Physical System Networks" by Peter Schneider and Konstantin Böttinger [10].

Challenge 3 mentioned in Section 1.1 also relates to the use of uncommon and proprietary protocols in CPS. As each manufacturer of devices may choose out of a variety of industrial network protocols, the inspection of network packets requires a thorough understanding of the used protocols and compatible processing strategies for each of them.

Contributions

To address this challenge, we propose a framework making use of feature-learning, unsupervised training, and parallelization techniques. We use a stacked denoising autoencoder to model an underlying distribution for industrial network packets. Given

the optimization goal of autoencoders to learn the identity function, we can train the framework without requiring to have anomalous network traffic at hand. Anomaly detection is then possible by applying thresholds to the reconstruction error of network packets processed by the autoencoder. We emphasize keeping the approach as lean and fast as possible to allow real-time processing of large amounts of industrial network traffic. Finally, we test the framework on two datasets using the Modbus and the EtherNet/IP protocols. As one of the datasets is only partly labeled, we propose a method for semi-automated labeling of such data to derive suitable quality measures.

In summary, we make the following contributions:

- We propose a high-performance processing framework for industrial network anomaly detection.
- We present a method for automated feature-learning for network packets independent of components and topology.
- We provide an approach for semi-automated labeling of unlabeled network traffic datasets.

5.1 Concept

Before developing a concept for anomaly detection, we describe the environment in which we aim to detect attacks and analyze how attacks might look like.

Current CPS are still static systems with a well-defined network architecture. While the application logic, i.e., which machine produces what, may change frequently, the physical and logical connections, i.e., which device acts as a master and which as a slave, do usually not change during production. Thus, the communication paths inside such a network are known in advance. Telemetry analysis can, therefore, be made as simple as enforcing the design specification of the system under test. The same is true for used protocols and the communication frequency between the machines. In contrast, the payload content of the network packets is directly linked to the physical process involved and is dynamic. While operators have the required insights to decode the transferred data, the interpretation of data as being good or bad is hard [32]. Most of the involved machines use proprietary or custom-developed protocols making deep packet inspection a hard problem.

An attacker might specifically look for causing damage to the machines, for disrupting the service, or for degrading the production quality of the CPS. Damage to or disruption of a service is easily achieved given physical access to the CPS by physically destroying components of it. The ever-increasing connections between machines and to Internet

or cloud services, however, make them vulnerable for remote attacks interfering with the system's availability, e.g., by distributed denial of service attacks.

Attacks targeting the behavior of the CPS already require a deep understanding of the used network protocols as a manipulation of the system without interfering with its availability needs to be compliant to the existing but closed system behavior [4]. Hence, we can assume that if an attacker is able to degrade the production quality of the system, he is also able to implement the network protocol correctly. Given these considerations, anomaly detection in this setting should focus on detecting changes in process-relevant control parameters, control timings, and relevant software behavior.

State-of-the-art anomaly detection systems need to be manually adapted to each CPS under evaluation to detect the described threats. The abundance of protocols, vendor-specific modifications, and requirements requires the support of CPS operators to acquire the needed system knowledge. To address this challenge, we describe a fully unsupervised learning-based processing pipeline. As we can apply our framework without in-depth knowledge and understanding of the relevant cyber-physical process, it is applicable to multiple threat scenarios, protocols, and environments without scenario-specific adaptations.

5.1.1 Anomaly Detection as a Classification Problem

Applying machine-learning to anomaly detection requires to model the task as a classification problem. We classify each network packet as either being benign, i.e., normal network traffic, or malicious, i.e., packets constituting an attack or being the result of an attack. After classification, we refer to malicious packets as *positive* and benign packets as *negative*. Each network packet p consists of $n_p \leq MTU$ bytes where MTU is the connection-specific maximum transmission unit, which equals 1500 bytes for Ethernet [100].

5.1.1.1 Padding

Let p_i be the i-th byte of the network packet p. As our later machine-learning method requires the inputs to be of equal length, we choose a maximum investigation size $MIS \leq MTU$. Packets shorter than MIS get padded by zeros while packets longer than MIS get cut off after MIS bytes. Using this approach allows for a trade-off between extracted information from the packet and the amount of data processed. However, MIS should be larger than the length of most of the network packets under test. The headers of network packets usually do not extend beyond the first 100 bytes

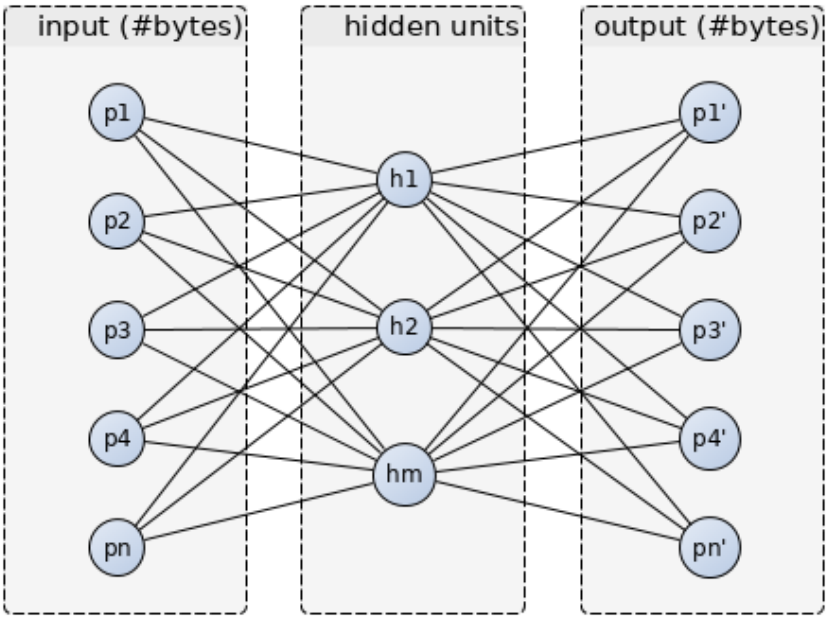

Figure 5.1: Schematic drawing of a single autoencoder layer.

and are contained in the sample as long as MIS is large enough. As most of the packets in industrial traffic are significantly shorter than the MTU, they more often get padded with zeros than cut off. This results in most of the information being used for the later anomaly detection. Thus, our input vector p^* equals to

$$p^* = \begin{cases} (p_1, \ldots, p_n, \underbrace{0, \ldots, 0}_{MIS-n}) & \text{if } n < MIS \\ (p_1, \ldots, p_{MIS}) & \text{if } n \geq MIS \end{cases} \tag{5.1}$$

5.1.2 Deep Autoencoders for unified feature learning and classification

As described in Section 5.1.1, the network packets are trimmed or padded to a specified input size. This enables fast preprocessing while omitting any feature extraction. We replace the usual step of feature extraction for machine learning applications by a feature learning approach based on current deep learning schemes. For this purpose, we consider each byte of a network packet being one value of the input vector p to an autoencoder network.

An autoencoder is a fully-connected neural network trained for replicating the input values as its output [101]. The network consists of three layers: one input, one output, and one hidden layer. While the input and output layers need to be of the same size, the size of the hidden layer can be adapted to the use case, cf. Figure 5.1. When denoting the values in each node of a layer as a vector, the values of the hidden layer can be written as

$$h = a(W_e p + b_e) \tag{5.2}$$

where W_e is a matrix containing weights for each link between every input and hidden node in the encoding stage, b_e is a bias vector, and a is a non-liner *activation* function wrapping the linear part inside. As we focus on constructing a high-performance pipeline, we use hidden layers with fewer nodes as in the input layer. This reduces the memory and processing requirements as fewer weights need to be stored and optimized. Thus, this encoding stage of the autoencoder maps the input to a smaller representation in the hidden layer. Using a decoding stage, the hidden representation is then transferred back to the original vector size with

$$y = a(W_d h + b_d) \tag{5.3}$$

Assuming y is similar to p, the hidden representation contains a lossy compression of the input p as it can be reconstructed using Equation 5.3 from the smaller representation h.

To achieve this state, we optimize the weight matrices and bias vectors during a training phase. Using the squared error of the current state se and a suitable amount of training examples, optimization techniques like gradient descent can then be used to obtain the weight matrices and bias vectors.

$$se = |p - y|^2 \tag{5.4}$$

After this training phase, it holds true that $y \simeq p$ and h contains a smaller representation of the input layer.

Figure 5.2 illustrates the inputs (left side) and output (right side) of a perfectly trained autoencoder supplied with an example of 5 bytes. Whereas the bytes i_1, i_2, i_3 and i_5 are reconstructed correct from the hidden representation, we suspect that a manipulated byte will cause the hidden layer being unable to reconstruct it. Hence, byte i_4 differs from o_4 and the reconstruction error rises.

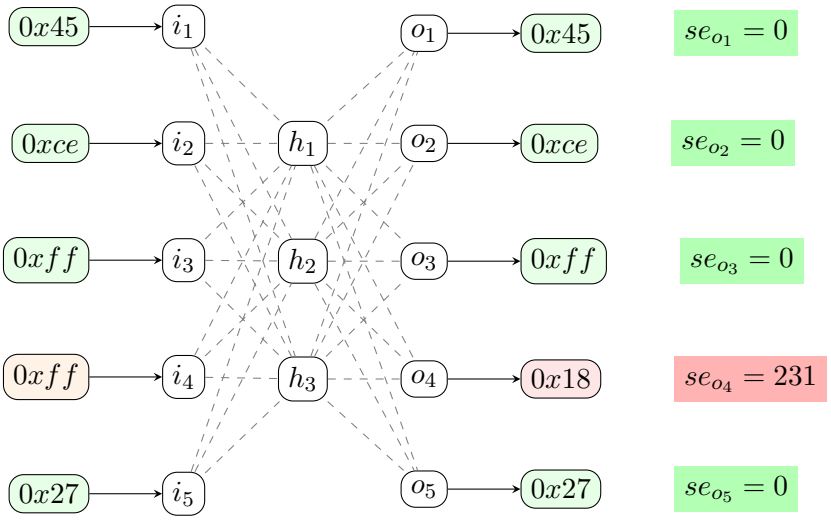

Figure 5.2: Example of inputs and outputs of an autoencoder.

Given this architecture and optimization goal of autoencoders, they are trained for deriving a low-dimensional representation of their inputs while compressing all information. Reusing the output of one hidden layer as a new feature vector for a second autoencoder results in an efficient feature learning mechanism. A separate feature extraction preprocessing step is not needed in this case. The introduction of stacked autoencoder layers is also referred to as deep neural networks [102]. Using the additional weight matrices and bias vectors, the model is capable of storing more information of the training data to reconstruct the input values.

In our anomaly detection framework, we stack two autoencoders, whereas each of them has a different number of hidden units, which is lower than the input vector size. The resulting architecture is shown in Figure 5.3.

The reconstruction error of a packet under this stacked autoencoder can then be written as

$$rmse(p^*) = \sqrt{(p^* - f(p^*))^2} \tag{5.5}$$

with $f(p^*)$ being the output of the autoencoder network.

Training a deep network like this results in learning a lossy compression scheme for the trained packets. As the stacked autoencoder is trained to reproduce the identity function, we assume that the reconstruction error, $rmse$, between the input and output vectors will be low if the packet under test is similar to the training packets and high otherwise. Putting limits on the reconstruction error then gives a classification method

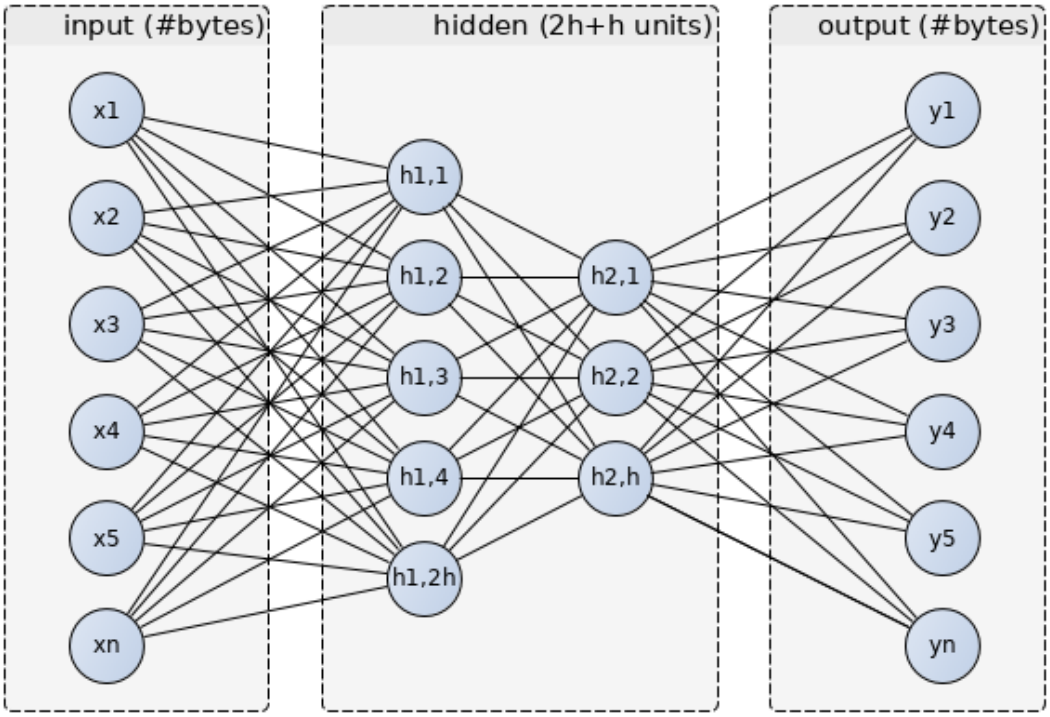

Figure 5.3: SDA architecture for anomaly detection on network packets.

in benign and malicious packets. Using the upper and lower thresholds t_{up}, t_{low} a single network packet p is classified by

$$p \text{ is } \begin{cases} \text{negative} & \text{if } t_{low} < rmse(p^*) < t_{up} \\ \text{positive} & else \end{cases} \tag{5.6}$$

During a cross-validation phase, we optimize the thresholds for the specific use case. The limits should be near to the minimum and maximum of the $rmse$ in the training data.

Like all deep neural network architectures, the application of stacked autoencoders may suffer from overfitting the training data. To minimize overfitting, we multiply the input bytes in p^* with random values of a normal distribution giving a modified input vector $\tilde{p}^*$ with some noise. During the training phase, we use $\tilde{p}^*$ as the input while we still compute the error and the training updates with the output vector of the Stacked Denoising Autoencoder (SDA) and the original vector p^*. This is a common approach for avoiding overfitting in stacked denoising autoencoders [103].

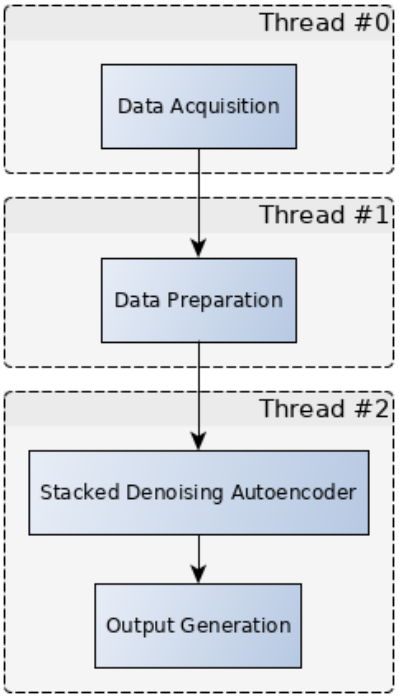

Figure 5.4: Parallelized processing using pipelining.

For later testing purposes, the original packet vector p^* without any randomized distortions can be used as the input vector.

This concept yields a feature learning and a classification method for industrial network packets in a single step.

5.1.3 Parallel Processing using Pipelining

As the processing of data in autoencoders can take up some time, we propose a parallel pipelining approach for faster processing. One technique for parallelizing the processing pipeline for big data analysis is the map-reduce approach [104]. A management system distributes tasks to worker nodes in a *map* operation and merges the results in the *reduce* operation. However, due to the focus on distributed processing, this approach is inapplicable in the context of confidential internal network traffic.

Since the usual training algorithms applied in deep learning frameworks work sequentially, parallelization can only occur between independent processing steps. In general, there are three processing steps required. First, network packets must be acquired either by reading packet capture files or by sniffing on network interfaces. To assure

that no packet is missed, especially when live capturing from interfaces, this task is wrapped in an own thread, as shown in Figure 5.4 (Thread #0). Second, the padding and length cut-off is part of the data preparation in Thread #1. Buffering packets and forwarding them in batches to the final processing stage allows for compensating the processing time needed by the autoencoder. As state-of-the-art deep learning frameworks most efficiently run on Graphical Processing Units (GPUs) [105], they always require transferring the data into specific memory on the GPU (Thread #2). To minimize these transfers and to maximize the processing efficiency, batch training is a common paradigm in these applications. Even after training, for testing new packets, batching is an advantage as it averages the scores over time and yields more stable results. Hence, we later use the $rmse$ of a batch of packets B defined as

$$rmse(B) = \frac{1}{|B|} \sum_{p \in B} \sqrt{(f(p^*) - p^*)^2} \tag{5.7}$$

5.1.4 Semi-Automated Label Estimation

A common problem in the development of anomaly detection systems is the unavailability of suitable training data. For industrial settings, this is especially true since the communicated data is always considered confidential and its leakage might threaten the competitiveness of its owners. If data is shared, it often contains unlabeled data since adequate detection and forensic tools are still unavailable making a correct labeling impossible. One of these datasets is the Secure Water Treatment dataset [68]. It is currently the largest industrial network traffic capture. However, it lacks the assignment of labels for the recorded network packets. Instead, we are provided with several packet captures for which we know whether they contain any attack or not per file.

While the application of the presented anomaly detection concept is possible without a packet-wise labeling, the method can only be tested regarding its detection performance with a ground truth available. Hence, to verify the concept's usability, we need a packet-wise labeling.

Figure 5.5 outlines a semi-automatic approach for deriving labels for a subset of the available network packets. In a first step, we apply our anomaly detection method to the data available, i.e., we train the described stacked denoising autoencoder unsupervised on the unlabeled training data. This yields a trained model that can be applied to new data (training phase). For each new batch of packets of the dataset to be labeled, we can now derive the reconstruction error $rmse$ given the Equations 5.1 to 5.7 ($rmse$ calculation). Sorting the batches based on their respective reconstruc-

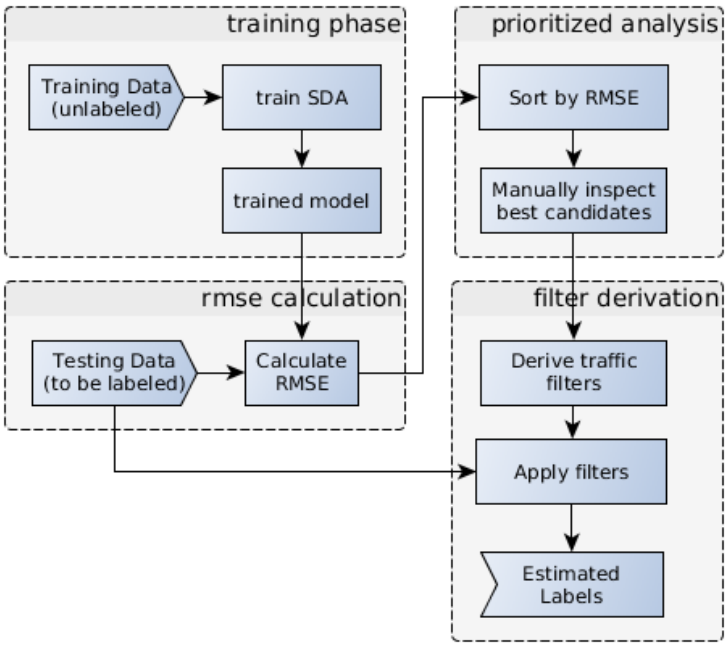

Figure 5.5: Semi-automatic estimation of traffic labels.

tion error gives a prioritization possibility for manual analysis of the highest anomaly candidates. We assume that by manual investigation of the batches with the highest reconstruction error and their communication contexts we can identify their class as being normal or anomalous traffic manually. For this purpose, network analysis tools, like Wireshark [106] or scapy [90], can be used (prioritized analysis). Once anomalous traffic has been found, specific filters or indicators for this type of anomaly can be created. Applying these filters to the whole testing dataset derives labels for the affected packets (filter derivation).

Using this approach, it is possible to label a portion of the dataset as being anomalous, i.e., all packets matching the derived filter are labeled. These generated labels allow for verifying how accurately a specific type of anomaly can be detected. Since all packets related to the attack type are labeled using the derived filter, we can match the detections from the framework with the corresponding network packets. This enables the derivation of the recall metric for this type of anomaly (cf. Section 5.2.1).

As only the most promising packets already indicated by the classification method will be analyzed, this cannot be an exhaustive labeling regarding further, less frequent attack classes available in the dataset. The calculated precision scores, therefore, represent a lower bound. If the dataset holds further attack classes not labeled by the filters,

these might increase the estimated false positive rate. A detection by the framework might be considered being a false positive because the packet is not labeled as anomalous due to a missing filter for the attack type. As the precision score (cf. Section 5.2.1) describes the ratio of true anomalies to the number of packets indicated anomalous by the framework, a too high estimate of the false positive rate results in a lower bound for the precision score. In scenarios with unlabeled datasets, the labels generated, thus, allow for validating the framework on part of the data regarding the precision and recall metrics.

The validation should be restricted to the attack classes derived using the approach outlined. As it is unknown whether there are further attack classes, the calculation of quality measures for the whole dataset is error-prone. Additional but not identified attack classes would lower the global recall values while they might increase the global precision values. As we should expect that not all attacks have been discovered, the lower recall values do not allow for a conclusive argumentation on the detection performance of the framework in general.

5.2 Implementation and Evaluation

We implemented the proposed framework in *python* using the *TensorFlow* framework [107] for processing. While we could have relied on data generated by our methods in Section 4.1 and 4.2, we decided to use two of the already published datasets [36, 68] to avoid being biased and benefit from the fact that Goh et al. [68] used a real-word system to gather a huge dataset. As we focused on implementing a real-time capable processing pipeline, we evaluated three network packet acquisition methods. Although packet parsing influences the processing time for each network packet negatively, it is included in most off-the-shelf tools for network forensics like python's `scapy` [90] module or the Wireshark [106] wrapper `pyshark` [108]. Comparing processing times for different amounts of packets (c.f. Table 5.1 and Figure 5.6) shows that the time increases approximately linearly to the number of packets processed. While the methods employing packet parsing, i.e., `scapy` and `pyshark`, need a similar amount of processing time, the `pcap` module [109] is up to 100 times faster. Since the first two methods extract specific flags out of the headers to disassemble the packets and rearrange them in object or JavaScript Object Notation (JSON) structures, they introduce an overhead for each processed packet. The `pcap` module, instead, directly forwards a sequence of bytes. In our evaluation datasets, using the packet parsing-based methods makes real-time processing almost impossible. For example, the recently published Secure Water

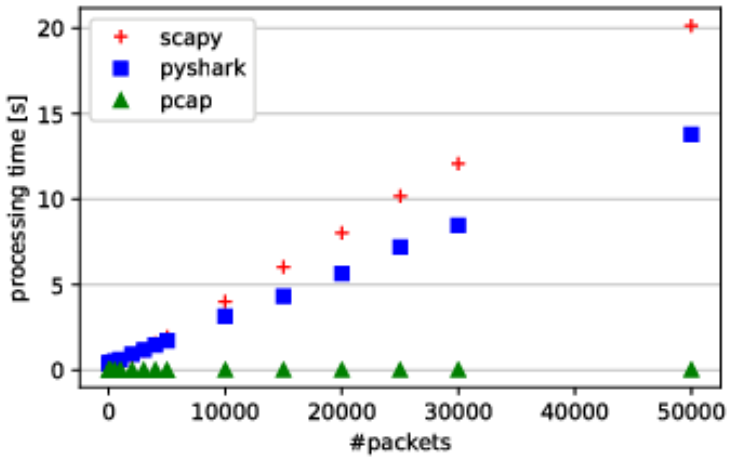

Figure 5.6: Performance of different packet acquisition libraries.

Treatment (SWaT) S3 dataset [68] contains network traffic with a rate of approximately $11M$ packets per hour. Given an ideal setup where the parallelized parts of the method do not influence the packet acquisition itself, this requires an acquisition rate of about 3056 packets per second. Such a high throughput is only achievable when omitting the packet parsing as the `pcap` module does.

In real-world applications, context switches and Random Access Memory (RAM) limitations may further decrease the performance especially for long-running captures. Further, common business network traffic consists mainly of the packet payloads (e.g., HTTP/S, FTP traffic) whereas the headers are much shorter than the actual payload. In industrial settings, operators try to enforce the shortest payloads possible resulting in many more individual packets when looking at the same network utilization. Considering a bandwidth of $100Mbit/s$ and assuming an average packet length of $100bytes$, a single captured network may see up to 131072 network packets per second. While using the plain `pcap` acquisition module the processing time can be expected to be less than $0.1s$, the parsing libraries will need more than half a minute.

As the framework itself does not rely on external packet parsing but instead infers required features by leveraging information in the hidden layers of the SDA, choosing the basic `pcap` module gives a significant speedup for the whole framework.

To restrict the requirements of working memory, we define the two autoencoders to use 20 (first stage) and 10 (second stage) hidden units with tanh as the activation function a and use a MIS of 1000 bytes.

# packets	scapy	pyshark	pcap
1000	$0.38s$	$0.63s$	$< 0.01s$
3000	$1.17s$	$1.19s$	$< 0.01s$
10000	$3.99s$	$3.15s$	$< 0.01s$
50000	$20.14s$	$13.79s$	$0.02s$

Table 5.1: Processing times for different amount of network packets in common network packet acquisition libraries. The times have been averaged over 10 repetitions on the same packets on an i7-4600U Linux machine @$3.3GHz$ and $8GB$ RAM.

After the training phase, the SDA can be run in a distributed setting using multiple worker nodes each one calculating the $rmse$ for one batch. Thus, the runtime performance of this stage is not as crucial as for the packet acquisition that needs to acquire and forward the packets sequentially and in time.

5.2.1 Deriving qualitative measures

Considering the definitions of *positive* and *negative* given in Section 5.1.1, the well-known precision, recall, and f1-scores can be used as qualitative performance measures. For this purpose, a *positive* classification result, i.e., the reconstruction error lies below the lower or above the upper threshold, is considered to be a *true positive* (tp) if the packet batch contained more than $n_{sig} = 2$ packets which constituted an attack or can be explained by an attack. Otherwise, the packet is considered originally being benign and the result is a *false positive* (fp), i.e., a packet which is benign was classified as being positive.

Correspondingly, *true negative* (tn) packets are correctly classified as being benign whereas *false negative* (fn) packets are classified benign while being related to malicious activity.

Precision denotes the ratio of packets being correctly classified as *positive* out of all reported *positive* detections. Thus, an anomaly detection framework with a high precision rate only raises an alarm if there really is an anomaly happening. *Recall* describes the ratio of anomalies detected out of all available anomalies. For a combined view of the performance, the harmonic mean of precision and recall, the f1-score, can be used. The described qualitative scores can be derived as

$$precision = \frac{tp}{tp + fp} \tag{5.8}$$

trace name	precision	recall	f1-score
run1_3rtu_2s	100.0%	100.0%	100.0%
exploit_ms08_netapi_modbus_6RTU_ with_operate	100.0%	87.5%	93.3%
moving_two_files_modbus_6RTU	100.0%	0.0%	0.0%
run1_6rtu	0.0%	100.0%	0.0%
CnC_uploading_exe_modbus_6RTU_ with_operate	100.0%	66.7%	80.0%

Table 5.2: Quality measures for the Modbus dataset according to the labels by [36] when applying thresholds $t_{up} = 2000, t_{low} = 500$.

$$recall = \frac{tp}{tp + fn} \tag{5.9}$$

$$f1 = 2\frac{precision \cdot recall}{precision + recall} \tag{5.10}$$

5.2.2 Modbus data

In a first experiment, we used the proposed method to detect anomalies in a dataset of Modbus network packets, which have been acquired in a simulation of a power grid control system [36]. The public dataset consists of 11 network traces which are all labeled. We used the `run1_3rtu_2s` trace to train the proposed SDA method as it is the largest packet capture available in the dataset that does not contain any attacks. For evaluating the detection approach, we tested all other traces that include attacks. The summary of quality measures in Table 5.2 shows mixed results. Basically, the approach either precisely detected the anomalies or not at all. To understand this behavior, we investigated the resulting $rmse$ from the SDA and the labels attached to the packets. The crosses in Figure 5.7 show the $rmse$ for batches of 200 packets in the `run1_3rtu_2s` trace, i.e., the $rmse$ on the training data, despite of a scaling factor. Figure 5.8 represents the error on the `exploit_ms08_netapi_6RTU_with_operate` trace during testing. The red shaded areas indicate where, according to the labels by [36], attacks happened. The color of the crosses refers to the resulting classification of the batch of packets. Batches shown as blue crosses lie between t_{low} and t_{up} whereas red ones have a $rmse$ of either less than t_{low} or greater than t_{up}. While the labels are available for every single packet, each cross represents a batch of 200 packets, i.e., all

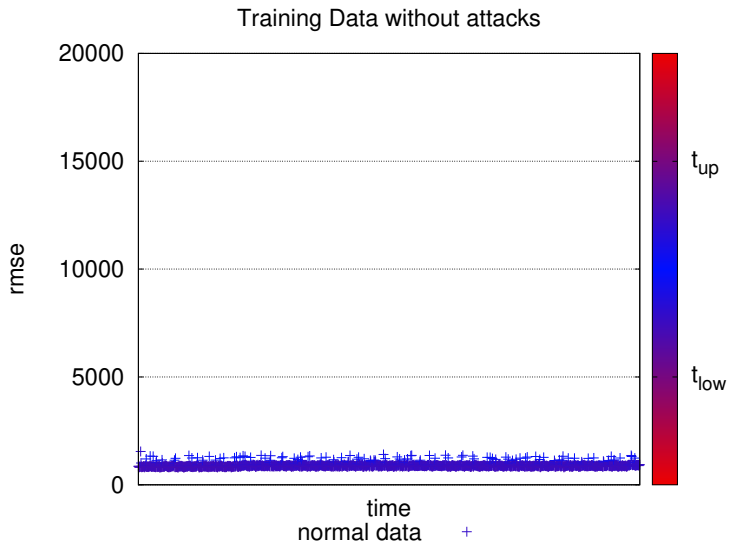

Figure 5.7: RMSE on the `run1_3rtu_2s` trace used for training on the Modbus dataset by [36].

packets starting directly after the previous cross up to the cross itself have the same error value.

The *rmse* during training (c.f. Figure 5.7) remains in a constrained band between ≈ 500 and ≈ 1000. To allow some outliers, we chose an upper threshold for anomaly detection of $t_{up} = 2000$ while we assume that the lower threshold can be chosen more strictly, e.g., $t_{low} = 500$. In Figure 5.8, it can be seen that the *rmse* rises, once the attacks started. The first data point still has a low *rmse* as in this batch only a minor portion of the packets is related to an attack. For the following seven batches, the *rmse* rises significantly over the threshold marking the corresponding batches as anomalies (red crosses). Considering the portion of anomalous-labeled packets in these batches which are indicated by the red shaded area, this behavior appears to be correct. With the last batch, the *rmse* then decreases again as there are no attacks in there. Using an upper threshold of $t_{up} = 1300$, attacks could be detected reliably in this trace.

Figure 5.9 shows the corresponding errors of packets of another trace. While in that case the *rmse* rises slightly after the attack times, the duration of the attacks is too short so that averaging in the batches keeps the error too low to detect anomalies reliably. Hence, the corresponding line in Table 5.2 shows a f1-score of 0%. We can, thus, conclude that there is a minimum duration required for attacks to be correctly detected using this approach.

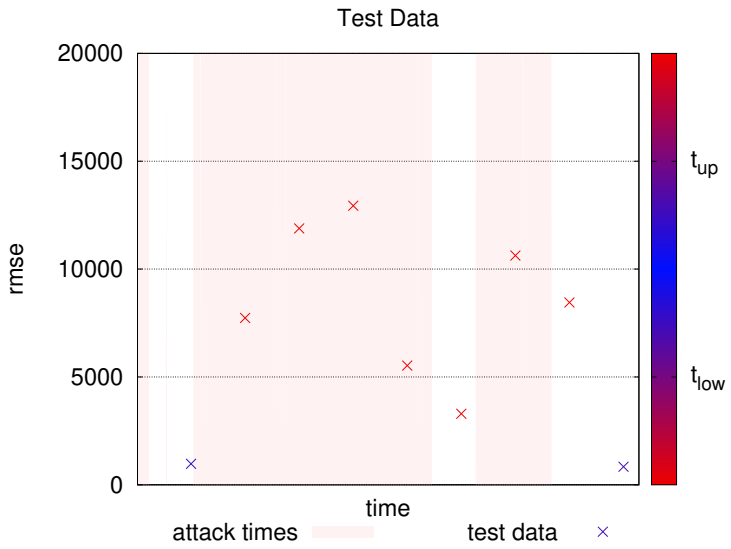

Figure 5.8: RMSE on the `exploit_ms08_netapi_modbus_6RTU _with_operate` trace.

In Figure 5.10 showing the *rmse* of the `run1_6rtu` trace, it seems like there are false positive detections yielding a precision score of 0%. Looking into the packet capture revealed unknown HTTP traffic at this time which was not present in the training data and which has been labeled as being benign by [36]. As the kind of traffic associated with HTTP connections is missing in the training data and the trained model, it is detected as an anomaly. Reconsidering these batches as being true positive detections, this trace would also come up to 100% precision and f1-scores.

5.2.3 Secure Water Treatment dataset

As the evaluation using the Modbus dataset showed that the approach suffers from too few data, our second test dataset consists of about $300GB$ of data collected during several days in the Secure Water Treatment (SWaT) S3 event [110]. The SWaT environment consists of several machines, programmable logic controllers (PLC), and computers, as well as 51 sensors and actuators which mainly use the Common Industrial Protocol (CIP) over EtherNet/IP (EN/IP) [68]. The data is split into two parts, one which does not contain any attacks and another one which does include attacks. However, as that data was collected during the S3 Capture-the-Flag (CTF) events, it is not labeled packet-wise and it is unclear what attacks have been carried out.

The first part of the dataset consists of 50 network traces, each including a roughly one-hour long capture with a size of about $6GB$. For the training phase, we use the

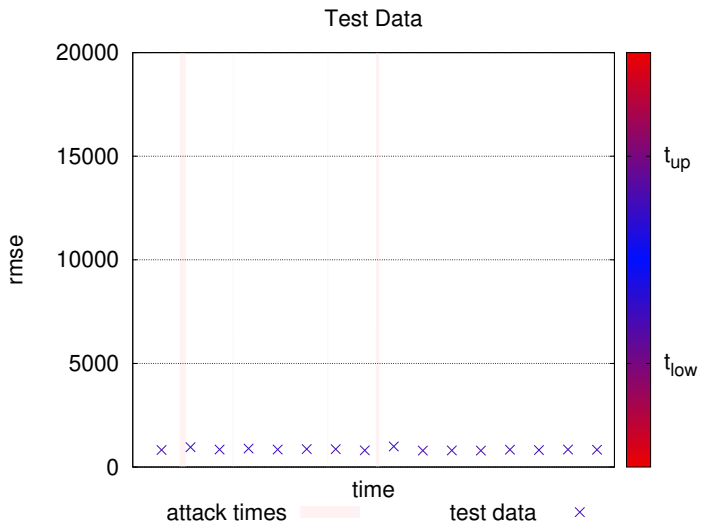

Figure 5.9: RMSE on the `moving_two_files_modbus_6RTU` trace.

`packet_00019_20170614105105.cap` capture, which contains 256,339 batches of 200 packets. As the underlying CPS is more complex, the portability and long-time applicability of the trained model is an interesting parameter to study using this dataset. We, therefore, tested the trained model against all available (not attack-containing) datasets available from the SWaT dataset. This includes data captured up to seven days later than the initial training capture trace whereas each trace itself is at least one-hour long. Figure 5.11 shows on the left side the *rmse* during the training phase. On the right side, the resulting *rmse* for all validation packet batches is shown, i.e., for 12,202,651 batches of 200 packets starting at 2017-06-14 11:51:05 up to 2017-06-22 16:04:08. Comparing the two figures shows that while the *rmse* is higher during the validation phase, it does not rise significantly. Thus, the model was able to capture enough information during training on one hour of traffic to perform similarly over several days. As the validation dataset is more than 47 times larger than the training set, we can assume that the model is not overfitted to the training data.

To test the detection capabilities of our model, we need to evaluate a dataset containing attacks and to apply the quality measures introduced in Section 5.2.1. For this validation, we use another trace from the later S3 event on the SWaT system, i.e., the second yet untouched portion of the dataset. This part comprises two network traces with a total size of $104GB$. The `s3171` trace starts at 2017-06-08 02:52:31, i.e., almost one year after the training traffic had been captured. For our anomaly detection evaluation, we use the first 200 million network packets of this trace resulting in 995727

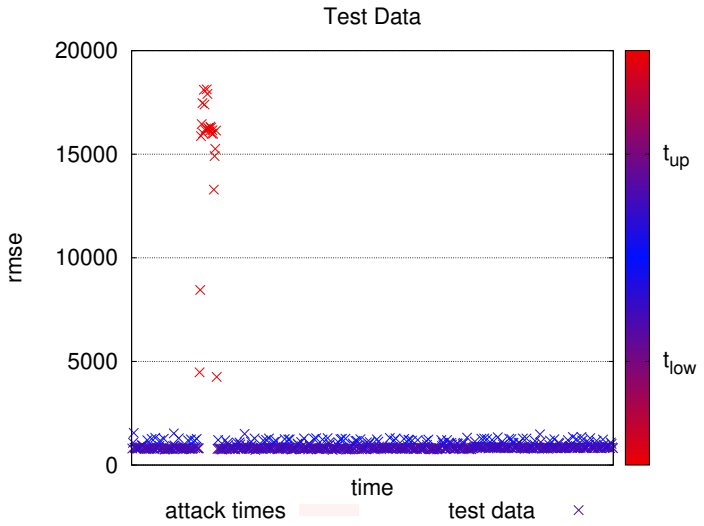

Figure 5.10: RMSE on the `run1_6rtu` trace.

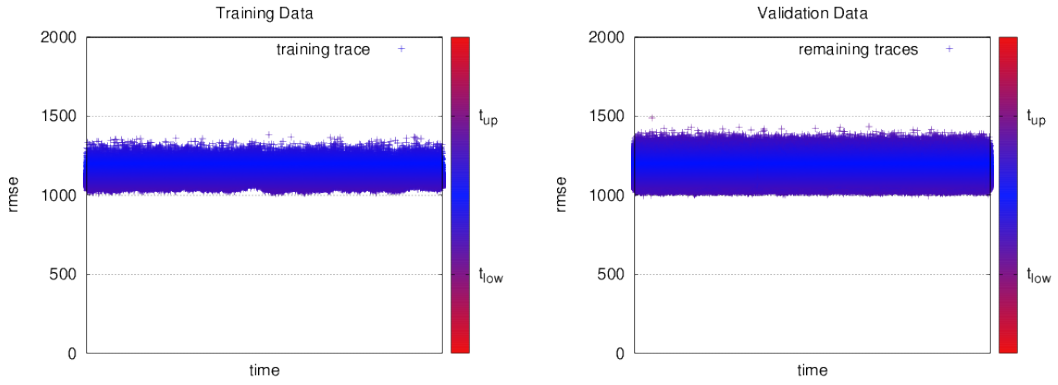

Figure 5.11: Root mean squared error during training and on validation data from SWaT.

packet batches. While we know this trace does contain attacks, it is unknown which packet should be considered benign or malicious as a packet-wise labeling is missing.

Nonetheless, we can process the trace and derive the corresponding *rmse* values for each packet batch. As shown in Figure 5.12, in contrast to the training and validation datasets (cf. Figure 5.11), the *rmse* occasionally exceeds the value of 1500 which had been an upper limit throughout the whole validation set.

5.2.3.1 Estimating Labels for Unlabeled Datasets

To derive labels for the second part of the SWaT dataset, we apply our labeling approach outlined in Section 5.1.4. This enables us to identify main root causes for

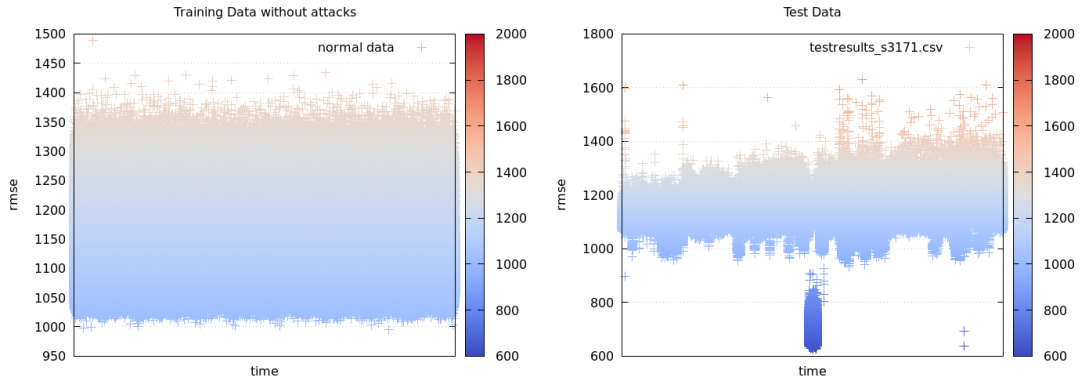

Figure 5.12: RMSE on the `s3171` trace.

anomalies appearing in the network traffic in the attack-containing part of the SWaT dataset. Our analysis revealed the following five major causes:

TCP Retransmissions of several packets indicate spurious network behavior. As retransmissions are used for lost TCP segments, there could be numerous causes. Frequent retransmissions might occur due to lost physical or logical connectivity or stalled services. Filtering those packets (*retransmit*) can be done with a scan on packet captures and identifying repeated TCP packets on a connection. In our implementation, we use the following Wireshark filter to identify this set of packets as being anomalous:

```
tcp.analysis.retransmission or
  tcp.analysis.fast_retransmission
```

Duplicate Acknowledgments are similar to TCP retransmissions, since they also indicate lost TCP segments. In contrast to the retransmissions, duplicate acknowledgments are triggered by the recipient to indicate the last seen segment. Testing whether a TCP packet has been acknowledged more than once can be done using state-of-the-art traffic inspection tools (*dupack*). In our implementation we use, thus, the following Wireshark filter to identify this set of packets as being anomalous:

```
tcp.analysis.duplicate_ack
```

TCP Resets are used to invalidate an ongoing TCP connection or to reject connections. These can be intentionally reused to interrupt services or to hijack connections. Thus, TCP resets may be part of real-world attacks. As such a reset is signaled

Line		dupack			retransmit			unknownproto-tls		
		p	n	$\sum$	p	n	$\sum$	p	n	$\sum$
1	t	3	974024	974027	1	973998	973999	2	973574	973576
2	f	44	73	117	44	99	143	44	523	567
3	$\sum$	47	974097	974144	45	974097	974142	46	974097	974143
4	pr		6.38%			2.22%			4.35%	
5	re		3.95%			1.00%			0.38%	
6	f1		4.88%			1.38%			0.70%	

Table 5.3: Anomaly detection performance as *pr*, precision, and *re*, recall, in problematic scenarios.

by a flag in the TCP header, filtering these packets is straightforward (*tcpreset*). In our implementation we use, thus, the following Wireshark filter to identify this set of packets as being anomalous:

```
tcp.flags.reset==1
```

Syn Flood Attacks are a method to disrupt the connectivity to a device by constantly sending handshake requests to arbitrary port numbers. Using our ranking method, we found several such attacks in the dataset and used a filter for TCP Syn packets which are not followed by a complete handshake (*synflood*) indicating connection attempts to dead endpoints. In our work, we implemented this with the following Wireshark filter to mark packets as being anomalous:

```
transum.status=="Response missing" and
  tcp.connection.syn
```

Unknown Protocols (TLS) in the network traffic are a strong indicator of anomalous actions going on. In CPSs, all required protocols are known in advance and can be included during the training phase. Thus, any newly occurring protocol may be part of an ongoing attack or other anomalous system behaviors (*unknownproto-tls*). In our implementation, we scan the training network traffic for all occurring protocols and use this list to compare any tested network packet. As this approach will list an abundance of protocols in the test dataset, we restrict this filter to only label packets as being malicious when containing a TLS layer. There is no TLS communication in the training dataset.

Using these five described filters, we derived labels for a portion of the network traffic. These labels can then be used for the calculation of quality measures.

Line		tcpreset			synflood		
		p	n	$\sum$	p	n	$\sum$
1	t	21528	974047	995575	21566	974094	995660
2	f	44	50	94	44	3	47
3	$\sum$	21572	974097	995669	21610	974097	995707
4	pr		99.80%			99.80%	
5	re		99.77%			99.99%	
6	f1		99.78%			99.89%	

Table 5.4: Anomaly detection performance as *pr*, precision, and *re*, recall, in well-working scenarios.

Evaluating the derived $rmse$ for each batch of 200 network packets against the thresholds $t_{low} = 900, t_{up} = 1500$ resulted in the observations listed in Tables 5.3 and 5.4. The first three lines show the distribution of packet batches being true or false and positive or negative. Lines four to six then list the resulting qualitative measures. For each label-filter, we count a true positive detection if the batch is positive and more than 2 packets in the batch matched the specific filter. A false positive detection is counted when the batch is positive but none of the label-filters matched. Hence, the number of false positives is equal across all filters. These are packets that we could not assign a clear root cause using our proposed labeling method. It may either be there are various smaller causes for these anomalies which require an understanding of the actual network traffic content or these are false positive detections by our framework. However, the other scores vary between the different anomaly sources outlined before.

Table 5.3 shows the quality measures for the detection of duplicate acknowledgments (*dupack*), TCP retransmission (*retransmit*), and unknown protocols (*unknownproto-tls*). The number of packets labeled as being anomalous can be derived as $tp + fn$. Looking at this value and the ratio of filter-labeled packets in the whole tested dataset, we can conclude that these anomalies are not frequent enough to be detected using the proposed method. While the *unknownproto-tls* filter matched 525 packet batches, this only represents 0.05% of the whole network traffic and is a too small portion of the network traffic. The other two filters matched even fewer batches. Remembering the findings from the Modbus dataset (cf. Figure 5.9 and explanation), this might also be a consequence of our batching method.

Considering the results of the more frequent anomalies shown in Table 5.4 further confirms this assumption. The ratio of filtered packets in *tcpreset* and *synflood* equals 2.15% and 2.17%. With the increased occurrence of anomalous network packets, the

detection performance rises to over 99% considering precision and recall in both traffic filters. This suggests that the proposed framework is a good solution to detect prominent network anomalies.

Another important finding can be seen in Figure 5.12. In the mid of the dataset, the $rmse$ drops significantly below the average value. According to our assumption, anomalies should increase the $rmse$. A manual inspection of these packets revealed that, in this case, there are syn flood attacks going on. The corresponding network packets mainly consist of their headers and no payload. Therefore, they can easily and precisely be reconstructed by the SDA as TCP SYN packets are part of regular network traffic. Due to the increased relative amount of SYN packets to other network traffic, the $rmse$ drops significantly. Thus, lower thresholds, as outlined in Section 5.1.2, can also be used to detect such shifts in network behavior.

5.2.3.2 Comparison against naive classifiers

Given the Equations 5.1 to 5.7, training a SDA for anomaly detection may lead to a naive classifier. As we pad shorter network packets with zeros, the corresponding input values for the SDA are 0. If, during training, most of the network packets are shorter than MIS, the weights of the connections from the last nodes in the input layer will not be trained at all, as they always evaluate to 0 regardless of the weights and biases chosen. Attacks utilizing longer packets than the average network packets during training, however, trigger these connections and may lead to an increased $rmse$. To test whether the SDA is nothing else than a packet length classifier, we use the same processing strategy but replace the SDA with a naive approach. During training, we calculate the average packet length avg_p over the whole dataset. When testing a batch of network packets B_i we use the average packet length of the batch avg_{B_i} and calculate the error e as

$$e = |avg_p - avg_{B_i}| \tag{5.11}$$

Using this error estimate, we can now use the same thresholding as a decision boundary. As during the training phase of the SWaT dataset, the error e remains strictly below 450, we use $t_{up} = 500$ for this test. Table 5.5 shows the results of this naive method for scenarios where the SDA-based method performed well. In these scenarios, the SDA-based method achieved precision and recall values of over 99% whereas the naive approach did not get one correct positive detection. This leads to an inferior performance compared to our proposed method. While using more statistical values the

Line		tcpreset			synflood		
		p	n	$\sum$	p	n	$\sum$
1	t	0	976754	976754	0	976772	976772
2	f	1638	21588	23226	1638	21570	23208
3	$\sum$	1638	998342	999980	1638	998342	999980
4	pr		0.00%			0.00%	
5	re		0.00%			0.00%	
6	f1		0.00%			0.00%	

Table 5.5: Anomaly detection performance as *pr*, precision, and *re*, recall, using a naive approach.

naive approach might be enhanced, the choice of those is restricted to easily acquirable information. Since parsing the packets requires too much time for real-time processing (cf. Figure 5.6 and Table 5.1), features like IP or MAC addresses cannot be used in a naive approach for the same setting.

Our evaluation showed that the framework is able to detect frequent and longer-lasting network attacks and outperforms more naive approaches. Considering the types of attacks we found during our analysis in the SWaT dataset, one might argue that there are easier and more reliable options to detect TCP resets or SYN flood attacks. However, we note that the framework achieves precision and recall values of over 99% without any information on the protocols during training. To correctly detect TCP resets, the reset flag (1 bit) in the TCP header can be read out. This information is not part of the training data or the model. Instead, the feature learning stage inside the SDA is able to capture enough information during training to detect these attacks. For SYN floods, one possible approach would be, for each machine, to mask packets going to ports, which are unused during training. While this information can be gathered in an application-specific adaptation phase, it is also not part of the training data.

We showed that the framework can be applied to different Fieldbus protocols without modification or adaptation despite the choice of problem-suitable upper and lower thresholds for detection.

The evaluation on test datasets obtained almost one year after the training data promises long-term applicability of a trained model as long as the overall system and network behavior does not change.

5.3 Conclusion

We present a method and framework for efficient and effective anomaly detection in CPS networks. During experiments, we show that our implementation of the framework is capable of achieving a throughput matching the requirements of industrial communication. By the omission of common network packet parsing, we achieve a packet acquisition speedup of up to 100 times. We overcome the resulting loss of interpretable features by applying a combined feature learning and anomaly detection method based on stacked denoising autoencoders. Finally, the evaluation showed that our presented method is capable of classifying typical anomaly indicators of high quality without initial information on the network or system behavior itself. Whereas we detect frequent and longer-lasting attacks with precision and recall rates of over 99%, shorter attack sequences may be filtered out by our batching approach. This suggests that with more advanced training and less batching, the detection performance may even be increased. Using the proposed method, we detected anomalies regardless of the specific industrial Fieldbus protocol. Without adaptation of the framework anomalies in the Modbus dataset [36] as well as in the Secure Water Treatment dataset [68] can be found. The ability to construct anomaly detection systems independent of the analyzed protocols enables broad applications in the currently evolving IIoT and CPSs. With our focus on processing vectors of raw bytes, the framework does not need to incorporate any parsing or information-byte mapping logic, making it suitable even for applications where such information is limited or restricted.

5.4 Summary

Contributions

With the presented anomaly detection method, we present several advances in anomaly detection method design. By objecting to deep packet inspection, we achieve a fast processing pipeline capable of handling also the latest CPS installations. Additionally, this frees us from protocol-specific restrictions, which may differ from scenario to scenario. By leveraging advances in machine-learning methods, we derive an unsupervised detection method from stacked denoising autoencoders. While this approach breaks with several common assumptions, we still manage to detect anomalies with very high confidence of over 99%. We even find further applications for our method in the semi-automated labeling of unlabeled datasets. This once again supports further research on future anomaly detection systems.

Referring to **C3**, we presented a high-performance method for unsupervised anomaly detection. This method bridges the gap between the accumulation of data from Chap-

```
1  [**] [144:1:1] (spp_modbus): Length in Modbus MBAP header does not
       match the length needed for the given Modbus function. [**]
2  [Classification: Generic Protocol Command Decode] [Priority: 3]
3  02/24-18:10:30.634900 192.168.1.104:502 -> 192.168.1.100:4386
4  TCP TTL:128 TOS:0x0 ID:38534 IpLen:20 DgmLen:57 DF
5  ***A***F Seq: 0xBF6CC890  Ack: 0xCF15D076  Win: 0xFADF  TcpLen: 20
```

Listing 5.1: Snort alert entry of a detected malformed Modbus network packet.

ter 4 to the following chapter, where we will further investigate the results of anomaly detection systems.

Relation to next chapter

Anomaly detection systems are often criticized for unintuitive resulting alerts. While the main goal of an anomaly detection system is to characterize the current system's state as either normal or not, IDS usually do not provide insights in the reasons for their decisions.

Alerts as the one from snort, a well-known business IT IDS, shown in Listing 5.1 only carry information on the state itself but not on the reasoning behind a decision as being anomalous. In addition, CPS operators are usually less interested in a single system's state but more in the overall CPS installations' state. Their main concern is to answer the question: "Can we still let our production systems running? Shall we intervene? What is the problem and how and where can we fix it?". To guide towards answers to these questions, the next chapter gives guidelines in IDS design to aid tracing of root causes and maximizing the overall benefit of IDS deployments in CPS.

6 Understanding Cause–Effect Relationships in Attack Campaigns

Purpose of IDS

Any security measure deployed should enable the system to strengthen its standing against future attacks. However, anomaly detection methods cannot prevent attacks themselves. The idea of anomalous state detection, instead, is to assist CPS operators and analysts throughout multiple stages of the security life cycle. Overall, detection systems are deployed to enable operators to understand the actual behavior of their CPS better as well as accelerating fixes for identified security problems before attackers may reach to critical systems. Most critical systems are not directly connected to the Internet but are protected by different security zones [75]. However, attackers may use vulnerabilities in less protected systems with remote connections.

Goals

Hence, the goal of this chapter is to identify root causes of detected anomalies to trace them back to their initial point of entry. Such an analysis allows for focusing protection measures to the locations where they are required the most. That identification eases further steps as locating the actual vulnerability and fixing it. Another common problem is the lack of understandability of the internal processes in modern intrusion detection systems. Hence, other security research domains already started investigating the relations of output alerts to their recipients' actions [111]. The next chapter introduces a formal model of distributed anomaly detection intending to make the interactions of complex installations more understandable.

Further, we combine this model with a simulation framework to derive recommendations for future detection installations in Section 6.1. This assists in cumbersome tasks like resource-constrained deployment and optimization to maximize the utility of distributed IDS setups, in Section 6.2.

With this understanding of distributed IDS setups, in Section 6.3 we finally provide a first concept for integrating multiple sources and methods into one coherent system. This system enables us to identify the current progress of an attack as well as its origin.

Parts of this chapter have previously been published in "Do's and Dont's of Distributed Intrusion Detection in Industrial Networks" by Peter Schneider [14]. The chapter "Optimization of IDS Deployments" is under review for publication and the chapter

"Understanding Advanced Attack Procedures in CPS from Heterogeneous Logs" is in preparation.

6.1 Simulation of IDS Deployments

With increasing system complexity and ever-rising new threat scenarios, researchers developed numerous tools and methods to derive anomaly and intrusion detection systems (IDS). Companies now implement these as steamroller tactics against yet unknown advanced attack scenarios. In case of detected attacks, anomaly and intrusion detection systems generate alert notifications. While the systems carry the potential of detecting new and complex threats, they are also prone to misinterpreting the current system state and consequently raising false alarms. The ratio between the detection of novel attacks and false alarms generated is usually a trade-off in the method design [112, 113]. However, many methods do not provide a guideline on how to choose this trade-off; instead, developers often employ experiments on real systems or parameter guessing. As experiments incur costs, they are usually limited in the explored parameter space. Hence, the derived parameters may be inadequate or sub-optimal. In consequence, operators face a significant amount of manual post-processing of final alerts if they desire high detection rates.

Often, the alerts are represented as entries in dedicated alert logs. Operators aggregate these files into knowledge databases using security information and event management systems (SIEM) and forensic tools. The entirety of the aggregated information contains valuable insights into ongoing attack strategies, present vulnerabilities, as well as indicators for future threat mitigation. The development of suitable countermeasures requires understanding the exact course of an attack strategy. Therefore, SIEM systems are often also used in forensic analyses and to recover from past attacks.

For such forensic analysis, it is fundamental to identify correlations and cause–effect relationships. Tracking down a sequence of them throughout a company network allows for a thorough analysis of the attack strategy and, especially, helps in finding the initial entry point of an attacker.

In practice, however, operators usually face a heterogeneous collection of alert log types, formats, and information gain making such an understanding time and resource-intensive. In their essence, all these log files represent a list of unsynchronized entries whereas the only commonality is the presence of a timestamp. Any further information in each entry is application-specific. The alerts may comprise a specific vocabulary or

even a technical encoding requiring dedicated knowledge to understand them. Forensic analysis of these files, hence, requires much manual investigation.

In this chapter, we provide a model for understanding cause–effect relationships between log files generated by intrusion detection systems in a compound system network. We show the interaction of these during an attack analysis and derive estimates on performance requirements for each detection system. Using a simulation, we are able to analyze the interplay of the systems and their joint performance in understanding ongoing advanced attack strategies. This allows us to set the detection and false alarm rates of an IDS into the context of the attack propagation speed and the underlying network topology. Finally, this analysis supports reasonable decisions on the parameters defining a detection method's trade-offs.

In summary, in this chapter we demonstrate

- how to aggregate information from different intrusion detection alert logs in Section 6.1.1,
- a simulation framework for understanding the interplay of different system parameters and resulting alerts in Section 6.1.2,
- and an analysis of different parameter configurations in Section 6.1.3.

6.1.1 Modeling Distributed Anomaly Detection

We start by providing a model for distributed anomaly detection on different abstraction levels. With this, we do not aim to provide a new anomaly detection method but rather to understand the general characteristics of existing and future methods. Second, we outline a simple mechanism to identify attack paths in a network of systems. Together, the anomaly detection model and the attack tracing mechanism allow for a study of the interplay of both concepts in Section 6.1.3.

Before detailing the actual model, a thorough definition of the used terms shall confine the scope of the method.

System A *system* may represent different parts of software in one device, different devices in a network, or even functions spread across different locations or organizations (see the following section). One system always corresponds to a single entity and can have multiple connections.

Anomaly Detection Method We understand an *anomaly detection method* as a function which, given a system state at a specific time, may detect anomalies. This

alert is either raised because the corresponding system is in an anomalous state or as a result of a false-positive detection.

Distributed Anomaly Detection We call a deployment of anomaly detection methods *distributed* if there are multiple systems involved. Further, the deployed methods may be heterogeneous or homogeneous. Thus, we do not require the same method for every system. However, we assume that all deployed methods are capable of detecting the same set of anomalies.

6.1.1.1 System and Connection Model

The term *system* and their corresponding *connections* can have several interpretations. In the remainder of this chapter, we only refer to these terms while the results and methods are applicable to all notions equally.

While a system resembles a single entity in our network, they interact via connections. This abstract definition allows us to apply our model and analysis to different abstraction levels. Systems can represent a dedicated host, physical device, or a machine in an IT network. These, then, may connect via Ethernet, WiFi, or the Internet. The only requirement for any link to be considered a connection is its capability of allowing an attacker to abuse it. Hence, this can also be extended towards USB devices used on several systems, or other forms of information transfer like the same user operating different machines.

Further, a system may also represent different parts of software on a single machine. A connection, again, is available if different software parts may exchange data. This may occur by accessing the same files, by using the same random access memory (RAM), by local communication on loopback devices, or by inter-process communication.

In a more abstract sense, we can also define a system to be any function carried out throughout a process or a whole organization. In this case, the definition of a connection becomes more difficult as there are various possibilities for information to be exchanged. However, the following methods would still apply provided our basic assumptions concerning attackers, their goal, and their methods still hold (cf. next section).

A similar concept is known as attack graphs. These describe an attacker's possibilities to tinker with assets. While our model is also graph-based, we do not consider these vulnerabilities directly. Whereas attack graphs are usually used during risk analyses to identify and later fix a system's weaknesses, we focus on choosing the right detection

measures with the right configuration. In contrast to attack graphs, we do not model system vulnerabilities but their behavior once a multi-stage attack started.

6.1.1.2 Attacker Model

Based on recent large-scale attack campaigns hitting Internet of Things (IoT) devices, like the Mirai botnet [64], we assume that an attacker wants to spread his attack to as many reachable devices as possible. Apart from this, gathering information about all reachable devices is usually part of the *reconnaissance* stage of cyber kill-chains [114]. Analyses by [3, 5] showed the presence of these in recent attacks on CPSs. Further, we assume that every device has suitable vulnerabilities for an attacker to access or manipulate it because systems used in Operational Technology (OT) for CPS usually do not provide security measures.

Concerning the whole network of systems, we envision a single attacker uses only one entry point to the overall system network. In reality, other systems may also be exploitable remotely. However, we assume an attacker aims for the probably easier attack propagation from systems already inside the target network.

In practice, multiple attacks from different attackers may spread simultaneously throughout a network. However, we assume that different attacks do not interact and, thus, do not alter a single attacker's behavior. Hence, without loss of generality, we only model a single attacker at all times.

In the following, we detail two exemplary use cases of our modeling framework. First, we use local alert log files on one host to understand the traversal of attacks throughout that host. Hence, we call this intra-host attack propagation and tracing. Second, we extend this concept to attacks targeting networks of computers.

6.1.1.3 Intra-host Anomaly Detection

Considering a single host, several types of logs provide information for anomaly detection. Application layer logs may carry information on anomalous events concerning respective functions. General system logs indicate the overall system status and allow for the identification of low-level anomalies and attacks. Additionally, each communication interface may be equipped with a logging or detection layer informing about anomalous network activity. Figure 6.1 illustrates this setup for two hosts. Regarding one of the hosts, there are three possible categories of attack propagation paths:

First, a remote attacker uses a vulnerability in the application logic to manipulate the application behavior. Hence, the application log records an anomaly. This manipu-

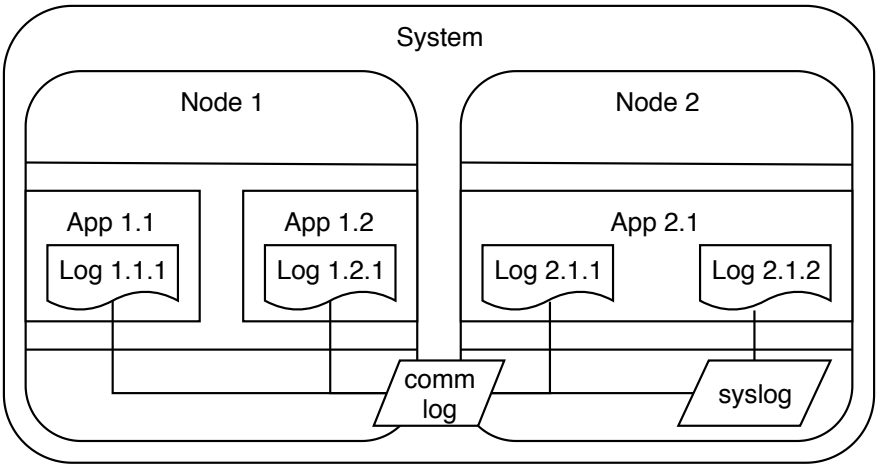

Figure 6.1: Localization of available alert log files and their relations to each other.

lation then possibly allows for further attacks on the underlying system. The attacker, hence, may influence the system state and extend its privileges to propagate to other hosts using available communication interfaces. The IDS on the communication interface, then, detects the anomalous network traffic.

$$application \rightarrow (system) \rightarrow communication$$

Second, an external network attack, like a man-in-the-middle packet manipulation or a physical attack on the network, targets the communication layer. This, in turn, influences the system's processing of the communication and leads to a manipulated system. After that, all applications on this system are subject to further manipulations of their application logic.

$$communication \rightarrow (system) \rightarrow application$$

Third, a direct attack on the system hardware or other local attacks may directly influence the system state. With a compromised system, an attack can further be propagated to an application or to other systems using the communication layers.

$$system \rightarrow application|communication$$

In the context of anomaly detection, it is important to note that attack propagation is possible without transferring the underlying system into an anomalous state. This can be done, for example, by crafting malicious network packets that comply with the

specifications but carry malicious payloads. Hence, logs for the system state may be less important for detection.

While we described these intra-host paths on an abstract level, they build the basis for every attack on any system. Studies like [3, 4, 5, 115] showed that real-world attackers chain these paths to multi-stage attacks. To detect such attack chains, we assume that subsequent detections on neighboring model parts correspond to the source and destination of an attack propagation. As they form cause–effect relationships, the order of detections in the different associated alert log files in the outlined attack propagation paths suggests their corresponding attack category.

6.1.1.4 Inter-host Anomaly Detection

In contrast to our simple setup in Figure 6.1, real-world network systems usually have much more independent hosts in different topologies. Therefore, we also need to investigate attack propagation characteristics in larger networks.

For the analysis of distributed installations of anomaly detection systems in larger networks, we consider an architecture $G = (V, E)$ comprised of systems V and connections E. Each system $v \in V$ is considered as a node of an undirected graph whereas a connection between v_i and v_j corresponds to an edge $e_{i,j} = (v_i, v_j)$ in that graph.

When modeling real-world systems, a connection may include systems on the same network, systems that are used by the same persons or which are physically close to each other. This way, we can also model attacks crossing air-gaps, like USB attacks [115], as well as attack vectors using social engineering.

The state of the system v_i is described by the two events *attacked*, $A_{i,t}$, and *detected*, $D_{i,t}$, at time t. Further, every system in the network may have an anomaly detection system $a_i = (d_i, f_i)$. This detection system is characterized by its detection and false alarm rates d_i, f_i. The detection rate indicates the probability that the system detects an ongoing attack at the system so that

$$d_i = P(D_{i,t+1}|A_{i,t}) \tag{6.1}$$

Accordingly, the false alarm rate is the probability of a reported attack while there is currently none so that

$$f_i = P(D_{i,t}|\neg A_{i,t}) \tag{6.2}$$

Additionally, we define the set of systems with reported detections at time t_k as $\mathcal{D}_k$. Given these definitions, we classify the detection of each anomaly detection systems as a

- true positive, tp, if $D_{i,t_1} \wedge A_{i,t_0}$
- true negative, tn, if $\neg D_{i,t_1} \wedge \neg A_{i,t_0}$
- false positive, fp, if $D_{i,t_1} \wedge \neg A_{i,t_0}$
- false negative, fn, if $\neg D_{i,t_1} \wedge A_{i,t_0}$

Using the sums of the detection classifications of all systems in the network, we can establish a general performance evaluation of the anomaly detection systems deployed in the network using the precision and recall metrics as

$$precision = \frac{tp}{tp + fp} \tag{6.3}$$

$$recall = \frac{tp}{tp + fn} \tag{6.4}$$

As these metrics refer to the average performance of the nodes' anomaly detection system, we refer to it as the *node precision* and *node recall*.

To model the propagation of attacks through the network, we assume that a compromised system s_i propagates attacks to adjacent systems, $adj(s_i)$, with the attack propagation probability i_i in two subsequent units of time.

$$i_i = P(A_{j,k} | A_{i,k-p}) \tag{6.5}$$

$$\text{with } p > 0, \quad \forall s_j \in adj(s_i)$$

$$P(A_{j,k}) = \sum_{s_i \in adj(s_j)} \sum_{t=0}^{k-1} i_i \cdot P(A_{i,t}) \tag{6.6}$$

Formulating this attack propagation reveals that the probability of a system being compromised rises with its exposure to other compromised systems. This exposure rises over time as well as with the system's degree in the graph. The degree k of a node n in a graph is defined as $k = |adj(n)|$. Hence, networks with higher connectivity, i.e., a higher average degree, should see faster attack propagation.

With this model and based on the previous cause–effect relationship assumption (cf. Section 6.1.1.3), it is possible to identify a likely path from newly attacked systems back to any previously affected system. A method for such a tracing procedure based on dynamic programming is shown in Algorithm 3. For each newly attacked system v

Algorithm 3: Tracing back new attacks in the network based on cause–effect relationships.

Input: Log $\mathcal{L}$, System graph G
Output: Set D of attacked systems

```
1  k ← 0;
2  D ← ∅;
3  foreach t, e ∈ L do
4      if t > k then
5          D_k ← D;
6          k ← k + 1;
7      end
8      if isDetection(e) then
9          v ← getSystem(e);
10         D ← D ∪ {v};
11         P ← shortestPathToSet(v, G, D_{k−1});
12         D ← D ∪ P;
13     end
14 end
15 return D;
```

at time k, we search for the shortest path P in G starting at a node $a \in \mathcal{D}_{k-1}$, i.e., a node which was already attacked at a previous point.

Based on the cause–effect assumption (cf. Section 6.1.1.3), the cause of a detected attack should be one of its neighboring nodes. If, however, no neighboring node detected the attack, we assume that the nearest node with a detection is the real cause. While more sophisticated attack path tracing methods are possible, we use the shortest path criterion as an example of how a tracing mechanism, in general, influences the overall system detection capabilities in Section 6.1.3.

Using weights on the edges of the graph, we can extend our model to include a specific propagation probability along each edge. As prerequisites, we need three special methods for each utilized detection system.

First, `isDetection` returns true if the provided log entry resembles a detected anomaly.

Second, the method `getSystem` must return one node of the graph at which the current anomaly was detected. For this, we can refer to Figure 6.1, which localizes different types of alert log files in the overall system. In case of communication logs, we define the corresponding system as the one at which communication endpoint the log was captured.

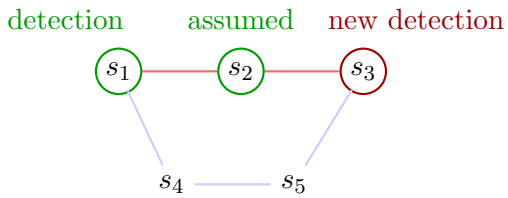

Figure 6.2: Illustration of assumed detections on shortest paths.

Third, the algorithm `shortestPathToSet` is a modified version of the Dijkstra algorithm [116]. The Dijkstra algorithm computes the shortest path from one starting node to all possible other nodes in the graph. In our adaptation, this set of nodes and path lengths is matched against the given set, i.e., $\mathcal{D}_{k-1}$. The shortest path from this subset represents the most likely path of attack propagation. We note that intermittent nodes on this path did not detect the attack themselves.

Consider the path $s_1 \to s_2 \to s_3$ describing the shortest path of a new attack on node n_3. As we only consider paths starting at already attacked nodes, s_1 must be attacked. If s_2 was attacked, however, the path $s_2 \to s_3$ is shorter as it is a subsequence of the first path. Hence, the path would already end at s_2's position. Therefore, we label all intermittent nodes on these paths as *assumed detections*. Figure 6.2 visualizes our concept of assumed detections on shortest paths. Without further knowledge it is also possible that the attacker actually used the longer path $s_1 \to s_4 \to s_5 \to s_3$. However, if we choose the required effort for an attacker as a distance measure between nodes, it is at least more likely that the shorter path has been taken. Such efforts can for example be derived in SRAs. The differentiation between *assumed detections* and other *detections* ensures that we do not alter the estimated attack paths on subsequent runs of Algorithm 3.

6.1.2 Implementation

In the previous section, we developed a model for aggregating and understanding information in different alert log files as well as tracing identified attacks through hosts and a system of those. Based on this model of information aggregation and attack tracing, we build a simulation allowing for a more controlled study of the influence of the different parameters.

While aggregating information is already part of deployed SIEM systems, it is yet unclear what the benefits of this aggregation are. Since the influences of different configuration parameters in a specific setup are still unknown, we need to understand the interplay of distributed information collection and centralized evaluation. Hence,

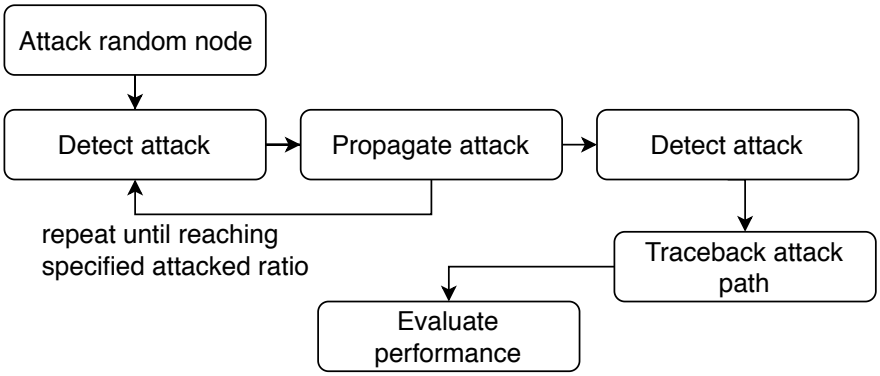

Figure 6.3: Process during simulations.

using the attack propagation and system network model proposed in Section 6.1.1.4, we implement a simulation tool allowing detailed experiments.

For this, we follow the process shown in Figure 6.3. We supply our simulation with a network model in the dot language of graphviz [117]. This allows for easy customization of the process to different network topologies and maps to our model of a system network as a graph. Each simulation run starts with an attack on one random node of the network. The rest of the simulation is based on a number of rounds: For each already attacked system and each of its connections, we now propagate the attack while respecting the attack propagation probability i_i. After the attack propagation through all systems, each of them uses its anomaly detection system and changes its detection state according to its detection rate, d_i, and its false alarm rate, f_i. A simulation round ends by incrementing the time counter by one and checking the ratio of so far attacked systems to the total system count. We repeat as many rounds until we reach a specified attack propagation ratio. After these rounds, we execute the traceback Algorithm 3 and derive the performance metrics (cf. Equations 6.3–6.4).

6.1.3 Experimental Results

In the following experiments, we use different sets of parameters to study their different effects. As we use Monte Carlo simulations in the simulation framework, we repeat every simulation setup with specific parameters for 100 trials. The Figures 6.7–6.14, therefore, always show the mean values of 100 trials. Aside from the parameters and their influence on the system behavior, we also test the influence of different network topologies on the performance. Hence, we test several network topologies relevant for current CPSs as well as industrial networks.

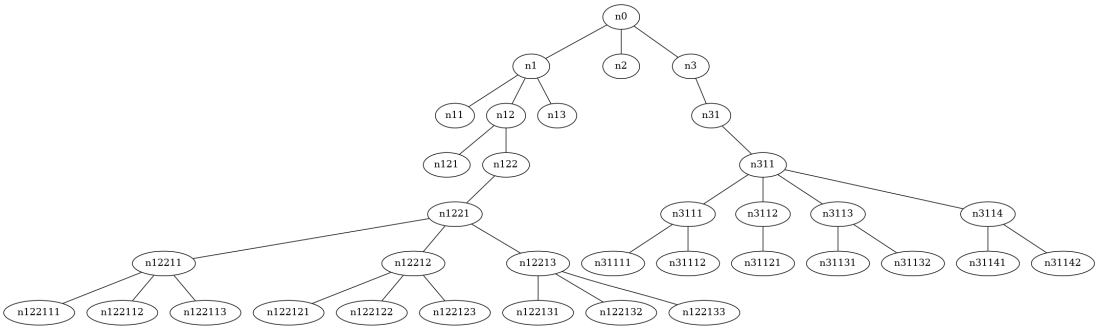

Figure 6.4: Hierarchical topology representing current networked CPS.

6.1.3.1 Topologies

As different network topologies may influence the performance of distributed attack detection, we choose three major network setups that may be encountered in current industrial and CPS installations. Ring topologies are often used in CPSs and the automotive industry. Our *ring* consists of 21 nodes and allows bi-directional communication (cf. Figure 6.5). Especially in distributed sensor networks, however, we often face *star* bus systems that use a central node for communication with several satellite systems connected to the central one (cf. Figure 6.6). In this setup, we have one central node and 20 satellite nodes. Recently, as part of the horizontal and vertical integration of CPSs, we move towards *hier*archical setups, as shown in our example in Figure 6.4. Also, the implementation of security standards, like IEC62443 [75], encourages the movement towards hierarchical setups. They are characterized by several zones which are connected using gateways. Figure 6.4 shows the setup we used in our hierarchical simulations.

6.1.3.2 Parameter Experiments

In the following Figures 6.7-6.14, we show average precision and recall values for all nodes and the overall tracing precision and recall. For the latter, we define a true positive detection of an attack propagation edge if once an attack was propagated along this edge in the detected direction. However, we do not consider whether Algorithm 3 made this detection at the right time. A true negative indicates that the algorithm did not output this edge and it was never part of an attack propagation. A false negative indicates that an edge used during attack propagation was not detected by any run of Algorithm 3. False positives indicate that Algorithm 3 once included an edge that was never part of an attack propagation step or was used in the opposite direction. By this

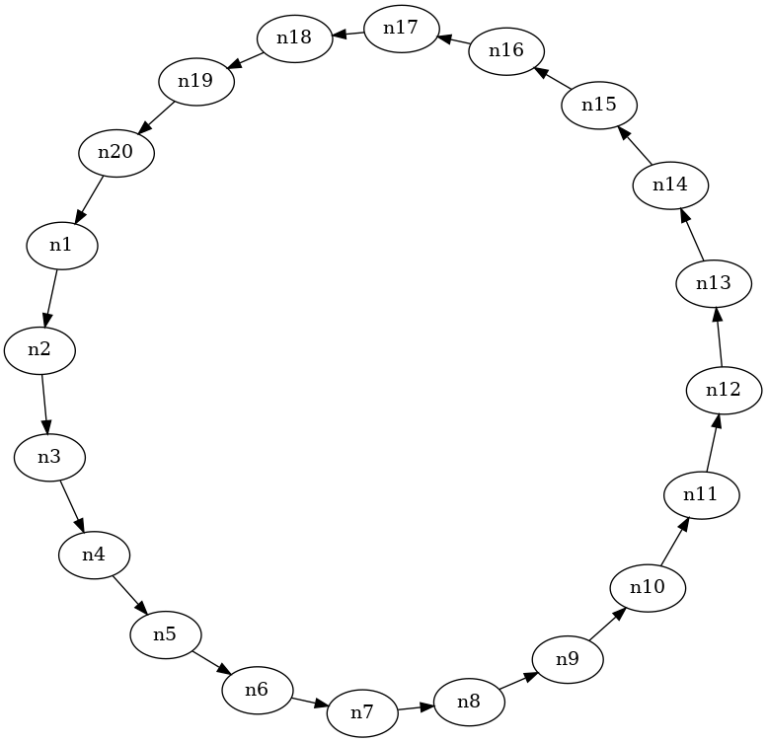

Figure 6.5: A classical ringbus topology.

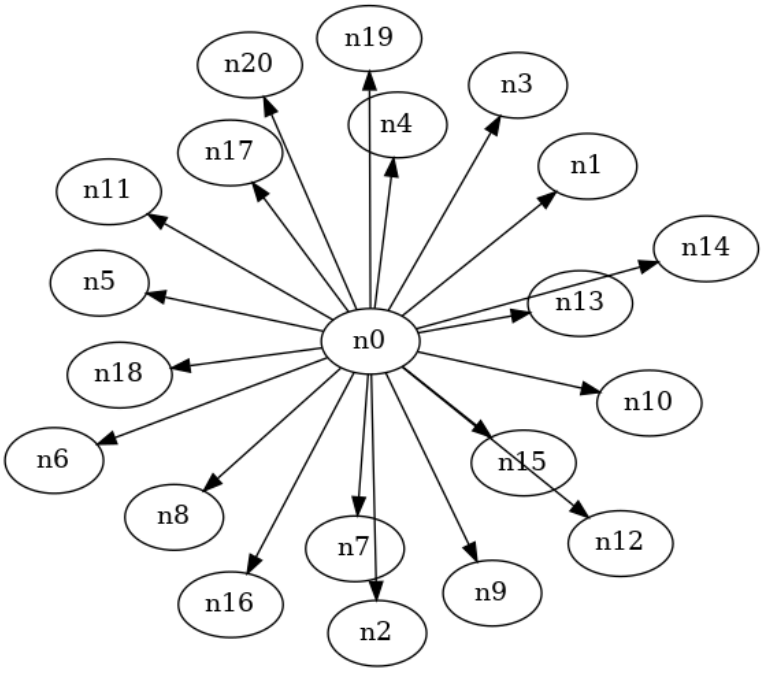

Figure 6.6: Star bus topology.

Detection in	precision	recall
SCADA communication [57]	>75%	>75%
plant data of a robot arm [54]	99.7%	100%
power system data [17]	88%	-
communication profile [118]	95%	97%
parsed network packets [119]	99.6%	99.6%
sensor values [120]	91.2%	86.1%
health sensor data [121]	95.7%	93.7%
SCADA network data (Section 5.2.3)	99%	99%

Table 6.1: Precision and recall values of recent anomaly detection methods for different data types derived from the indicated publications.

definition, it is possible to derive precision and recall metrics according to the definition in Equations 6.3-6.4. Analogously, we call these *trace precision* and *trace recall*.

For comparison, Table 6.1 lists precision and recall values of some recent CPS anomaly detection methods. When comparing with the following analysis, however, we need to remember that these methods are optimized for detection in specific application data. Our simulation, instead, is data-agnostic and studies the relationships between these parameters independent of their application domain.

Detection Rates In this experiment, we run the simulation with a false alarm rate of $a_i = 0\%$, an attack propagation probability $i_i = 70\%$, and a final attack propagation ratio of 70% while all systems have an anomaly detection system. Figures 6.7–6.8 illustrate the impact of the detection rate on the overall node and tracing performance. In the first figure, the graph shows the mean precision and recall values for nodes. The differently colored crosses demonstrate that the mean node precision is almost independent of the detection rate. With a false alarm rate of 0%, every detection from an anomaly detection system must be a real detection. However, the graph also shows values slightly below the 100% precision value. Those are due to possibly wrong detections based on the traceback algorithm. As noted in Section 6.1.1.4, we label intermittent nodes as *assumed detections*. For calculating the performance measures, these are included in the same way as the detections from the anomaly detection systems. Hence, wrong decisions in the traceback algorithm concerning the attack paths may introduce false detections for single nodes.

In contrast to the crosses, the triangles show a dependency between the detection rate and the ratio of attacked nodes detected. First, we note that higher detection rates always yield higher recall values. Second, we now see a difference in the performance of

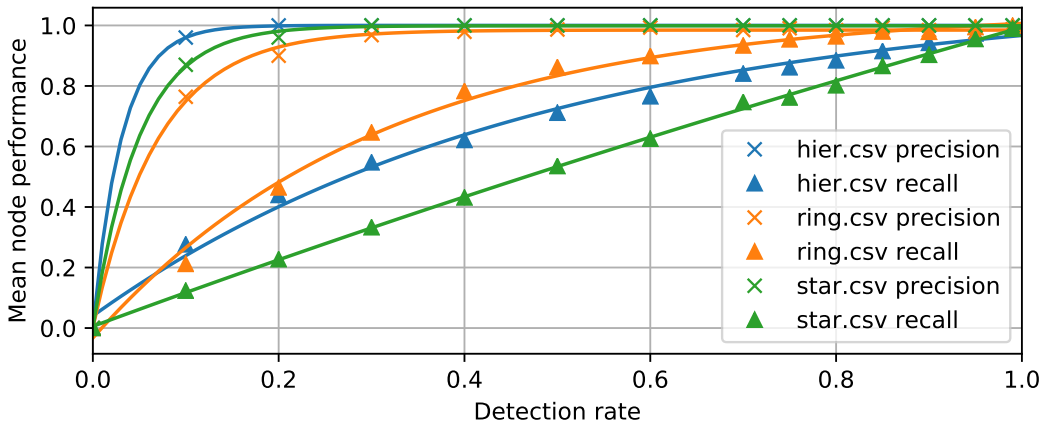

Figure 6.7: Impact of detection rates on node performance.

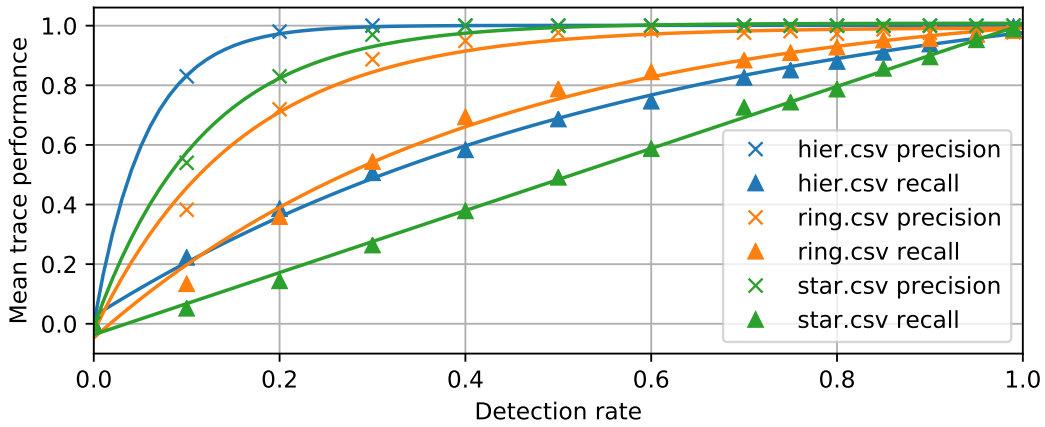

Figure 6.8: Impact of detection rates on trace performance.

the different topologies. In general, recall values for the ring topology are higher than for the star while the hierarchical topology is placed between them. With a detection rate of only 30%, the recall values for the ring topology are even twice as high compared to the star topology. With an increasing average path length, the tracing algorithm is capable of identifying more assumed detections. Hence, attacked nodes are identified earlier. The second figure shows similar results for the detection of attack paths.

False Alarm Rates Figures 6.9–6.10 illustrate the impact of the false alarm rate on the overall tracing performance. For this experiment, we run the simulation with a detection rate of $d_i = 90\%$, an attack propagation probability $i_i = 70\%$, and a final attack propagation ratio of 70% while all systems have an anomaly detection system.

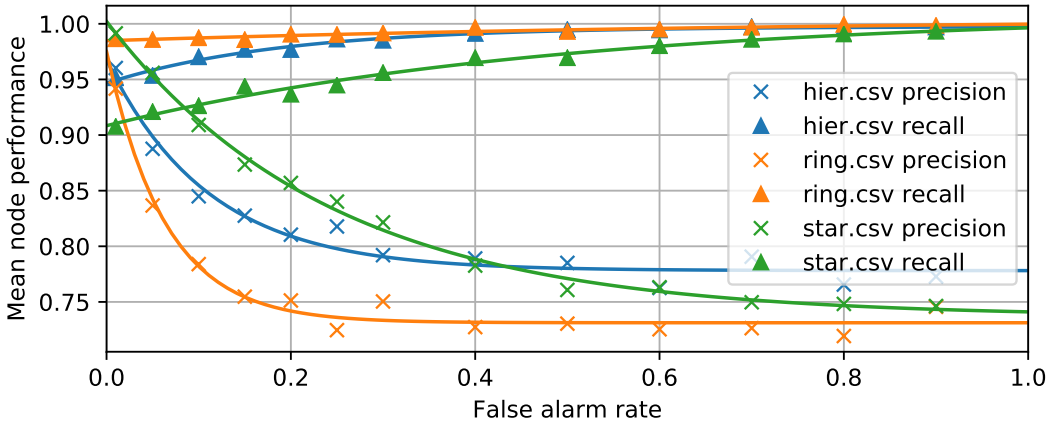

Figure 6.9: Impact of false alarm rates on node performance.

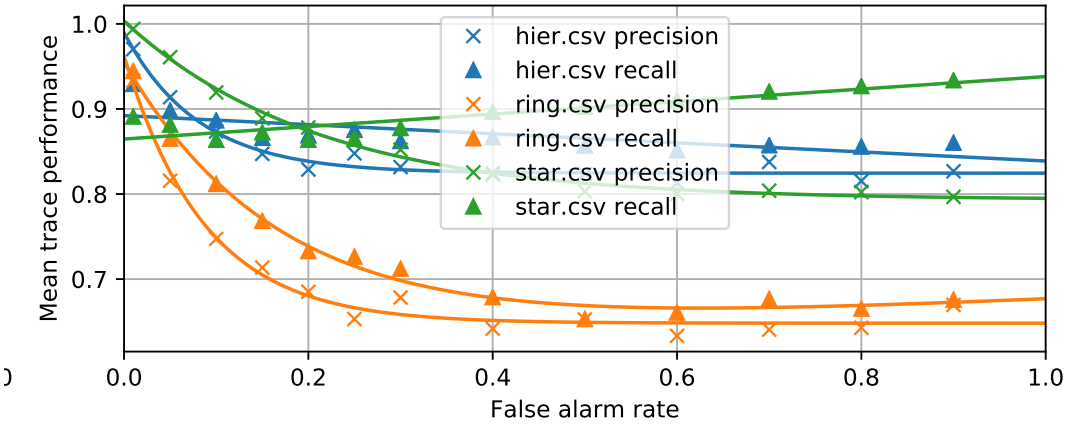

Figure 6.10: Impact of false alarm rates on trace performance.

The mean node performance can be described without differentiating the topologies. A higher false alarm rate leads to less precision in the average node detection performance. For the ring topology, while longer average paths helped in detecting attacks in the previous experiment, introducing more false alarms leads to worse performance. However, simultaneously the recall values slightly increase. With higher false alarm rates, the chance of every single node to signal a detection rises. Hence, finally every node signals a detection and, thus, also every attacked node does so. Therefore, we reach recall values of up to 100%.

For the attack paths, we see a similar drop in average precision. In contrast to the average node recall, the tracing recall does not necessarily increase. In the case

of the ring and hierarchical topologies, the recall values decrease almost in the same way as the precision values. The tracing Algorithm 3 heavily relies on the correct order of detections. If two detections are exchanged in their order, the resulting attack path has the wrong direction and will be considered a false positive in the performance evaluation. Once again, this effect is more pronounced with longer average path lengths. Additionally, for a star topology, this even yields increased recall values for the same reason as the node recall values increase.

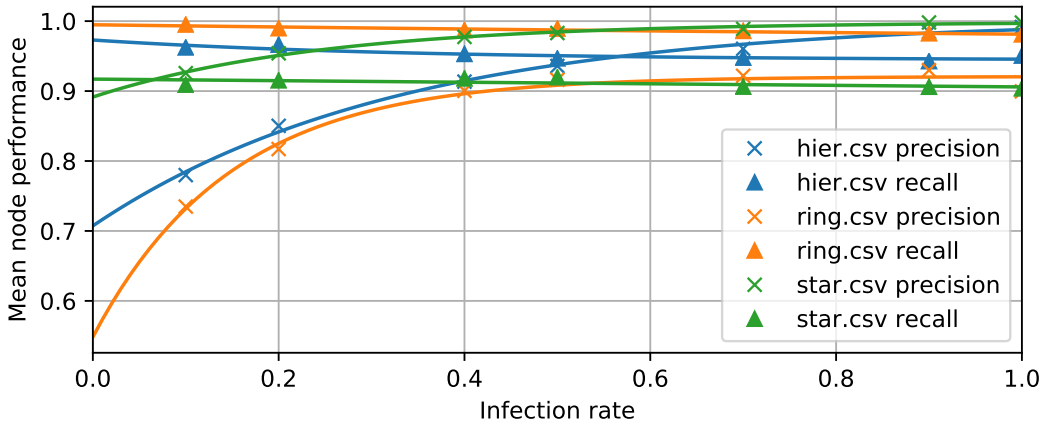

Figure 6.11: Impact of attack propagation probability on node performance.

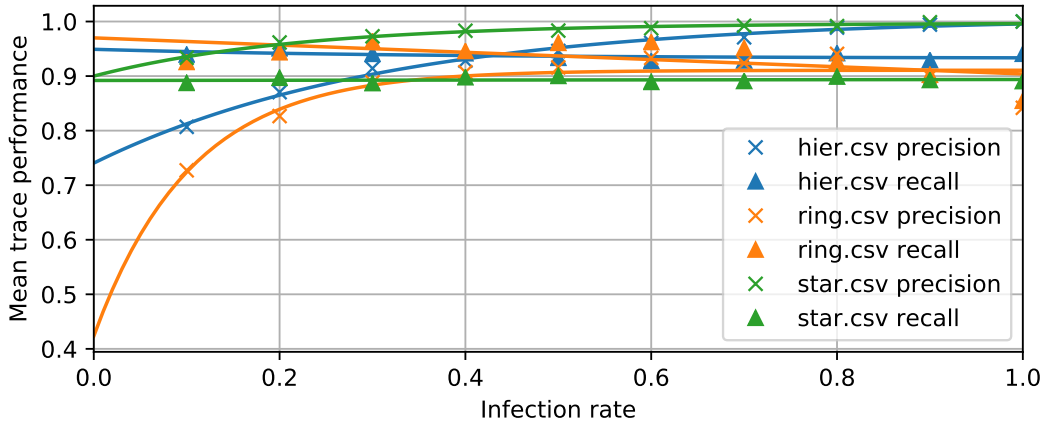

Figure 6.12: Impact of attack propagation probability on trace performance.

Attack Propagation Probability Testing the effects of the attack propagation probability, i.e., i_i, on the performance, in general, we see no significant difference between

the average node and tracing performance. For this experiment, we run the simulation with a detection rate of $d_i = 90\%$, a false alarm rate $a_i = 1\%$, and a final attack propagation ratio of 70% while all systems have an anomaly detection system. The precision increases with higher attack propagation probability (cf. Figures 6.11–6.12). With a lower attack propagation probability, the time for reaching the final attack propagation ratio increases. This, further, allows more false detections as a result of the false alarm rate. Hence, the faster the attack propagation the more accurate the detection, i.e., the higher precision values. The recall performance values change only slightly with different attack propagation probabilities. They do, however, show a slight decrease with increased attack propagation probability. This is caused by the same time window of detection opportunities as stated before. With a higher attack propagation rate, the time to detect an attack gets shorter and might be missed by an anomaly detection system. Only due to the high detection rate in this experiment, the decrease is small.

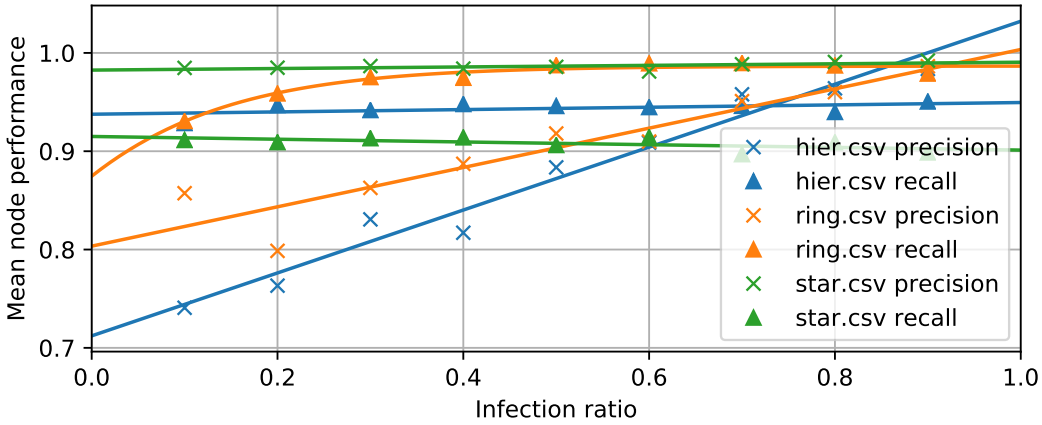

Figure 6.13: Impact of attack propagation ratio on node performance.

Attack Propagation Ratio For this experiment, we run the simulation with a detection rate of $d_i = 90\%$, a false alarm rate $a_i = 1\%$, and an attack propagation probability of $i_i = 70\%$ while all systems have an anomaly detection system. We alter the attack propagation ratio, i.e., the point when the simulation stops based on the ratio of already attacked systems. After the last simulation round, we still run one detection-only round to give every node the opportunity to detect possible attacks from the very last round. The recall values in Figures 6.13–6.14 do not significantly change during this experiment neither for the mean node nor for the mean tracing performance. However, for the hierarchical and the ring topology, we see an increase in the

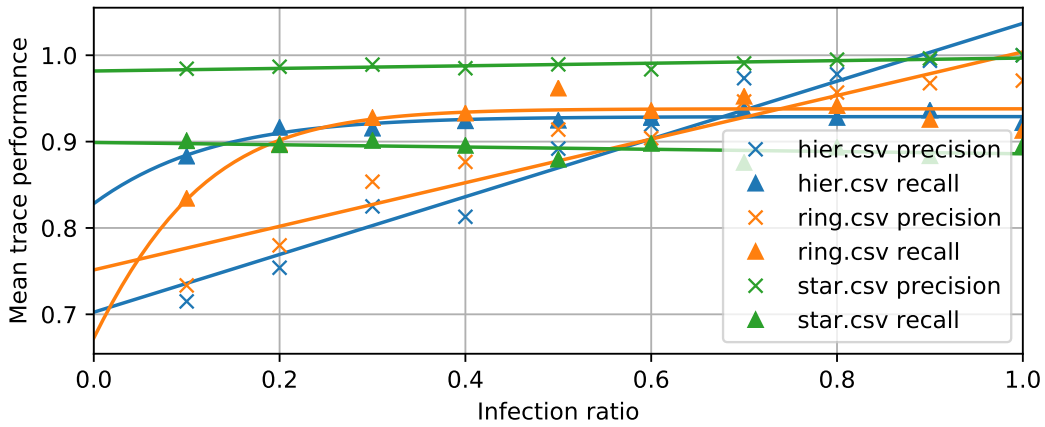

Figure 6.14: Impact of attack propagation ratio on trace performance.

precision with increased attack propagation ratios. Those two topologies have longer average path lengths. Therefore, they are also more prone to false *assumed detections* in consequence of the traceback algorithm. These may reduce the precision, especially when the tracing is done earlier, i.e., with lower attack propagation ratios. For higher attack propagation ratios, there is simply no false detection possible as almost all nodes have been attacked already.

IDS Distribution Ratio For this experiment, we run the simulation with a detection rate of $d_i = 90\%$, a false alarm rate $a_i = 1\%$, an attack propagation probability of $i_i = 70\%$, and a final attack propagation ratio of 70%. The results are almost the same as for the detection rates (cf. Figures 6.7–6.8). Like for the detection rates, with a decreased ratio of anomaly detection systems available the average probability for a node to detect an attack is reduced. Hence, we see the same behavior as with lower detection rates.

6.1.3.3 Parameter Effects

The goal of a parameter optimization is to reduce the false alarm rate, i.e., increasing the precision while achieving the highest possible attack detection coverage, i.e., the highest recall. Hence, the previous discussion of different simulation results leads us to the following recommendations for the deployment of distributed anomaly detection setups:

[Do 1] More IDS sensors for compact networks. The experiment on detection rates showed that longer average path lengths in a network might compensate for

lower detection rates. With the same detection rate, a ringbus or hierarchical network structure has a higher detection recall. Longer paths, however, increase the risk of misinterpreting attack paths and, in consequence, also lead to higher false alarm rates.

[Do 2] Better detection rates for fast-spreading attack strategies. While the attack propagation probability is a fixed parameter in real-world scenarios, it shows a direct link to the detection rate. Assuming we see fast attack propagation in a specific scenario, we, thus, should focus more on higher detection rates than on lower false alarm rates. Due to a decrease in the window of opportunity for detection, the effective detection and false alarm rates decrease. In our experiments, with attack propagation probabilities above 40%, the precision is almost identical to the detection rate. With faster propagation, the impact of false alarms is, therefore, negligible.

[Do 3] During forensics, use large lookback windows. The experiment on attack propagation ratios shows that the node and trace precision resulting from the traceback algorithm increases linearly with the attack propagation ratio. As using a large lookback window includes most previous attack steps and uses them as part of the path reconstruction, the tracing is more reliable. Another option is to fine-tune the traceback algorithm by restricting the maximum path length considered or by requiring a minimum amount of attacked nodes before execution.

[Don't 1] High recall values caused by high false-alarm rates. The experiment on false alarm rates clarifies that high recall values may well be caused by false alarms instead of detection capabilities. An IDS must, thus, always be judged comparing precision *and* recall values. For the methods based on machine-learning, this is usually done using the F1 score, the harmonic mean of precision and recall.

$$f_1 = 2\frac{precision \cdot recall}{precision + recall} \tag{6.7}$$

[Don't 2] Confuse low false alarm rates with fast attack propagation. As shown in the experiment on the attack propagation probability, low false alarm rates correlate with a fast attack propagation. If we face attack strategies spreading quickly throughout a network, the time window for false alarms becomes shorter. Hence, the false alarm rate tends to decrease. In reality, however, advanced attacks like Stuxnet [3] penetrated systems slowly. Thus, in this case, the same IDS system would trigger more false alarms as expected.

[Don't 3] Misinterpret high detection capabilities under worst-case scenarios. When facing scenarios of distributed anomaly detection where almost all participating systems are compromised, we must expect high detection rates. In this case,

every alarm raised by an IDS is a valid one as every node is already compromised. Hence, the false alarm rate decreases to 0% and the detection rates approach 100%. When comparing the capabilities of different IDS, worst-case scenarios are therefore less suitable.

6.1.4 Conclusion

In the previous sections, we outline a model for log file classification, anomalous event information extraction, and event tracing. Within this model, we localize different log file types and formats. Further, we develop a graph-based model of distributed anomaly detection systems. We use a simulation of this distributed system for an anomaly detection parameter analysis. In our analysis, we study the impacts of various configuration changes during simulations. While the recall performance in ring topologies profits from higher detection rates and higher attack propagation as well as IDS distribution ratios, the precision suffers from high false alarm rates and attack propagation probabilities. Star topologies, however, do not show a different network behavior compared to the single node performance. Both, precision and recall, profit from high detection rates and suffer from high false alarm rates. They are almost unaffected by the attack propagation probability and ratios as well as the IDS distribution ratios. Hierarchical topologies form a mixture between ring and star topologies. Hence, they show a combination of their results.

The observations in our experiments allow for more fine-tuned distributed anomaly detection setups. Taking into account different network topologies, we derive recommendations for specific use cases. For future research, we summarize these findings into seven design rules for distributed anomaly detection systems. While most anomaly detection methods provide a trade-off parameter between higher detection rates and allowing higher false alarm rates, a method for choosing this trade-off is often lacking. Our approach using Monte Carlo simulations supports this choice by a thorough analysis. With the provided model, choosing this trade-off evolves from guessing and experimenting towards a reproducible decision.

6.2 Optimization of IDS Deployments

With the presence of commercial-off-the-shelf (COTS) tools for the business IT sector, many operators search for deploying similar intrusion detection systems in their CPSs. However, the characteristic features available in business IT do not reflect the same information in CPS. In [32, 52], the authors explain that the inclusion of the physical domain provides additional information that is not sufficiently represented in the characteristics usually used in business IT IDSs. Therefore recently, new methods [10, 12, 37, 89, 118, 119, 122, 123] establish adapted intrusion detection schemes for CPSs.

However, while the adapted IDSs provide more reliable detection for CPS, they have similar resource requirements as business IT IDS. Equipping all nodes with an effective IDS enables high confidence in attack detection, false alarm detection, and attack vector identification. In contrast to the detection of known signatures, the recognition of anomalies requires high computation performance. Instead, as CPSs are often constrained by low resource requirements in terms of memory, computational power, energy consumption, or network bandwidth [8], not every system may be capable of deploying an IDS. Access- and organizational restrictions can further inhibit the use and deployment of IDS in this domain. To protect against faults and tampering, some devices are also locked down by their respective vendors. Although preventing unintentional or malicious changes, this again hinders the usage of IDS for CPS. Others may be restricted in terms of their computational or electrical power. Typically, the primary functions completely utilize the limited resources of the minimalist sensor nodes that are the basis of most CPS. Thus, there is no room for any additional anomaly detection processing. Finally, specific safety or security certifications can inhibit the inclusion of an IDS by requiring complete recertification due to the modification. Due to these obstacles, full coverage of all parts of a CPS is not a practical option.

Dealing with restricted resource availabilities always demands proper planning and distribution. For IT security, one measure for allocating the right capacities are security and risk analyses pointing out the specific needs of a system [12]. However, they cannot capture the interaction and interplay of several anomaly detection systems as it is available from distributed anomaly detection [69, 70]. Hence, in this work, we analyze the effectiveness of specific IDS deployments under constraints. Using this analysis, we can then infer optimized IDS deployment locations. We suggest different optimization goals and methods providing a framework for optimized IDS placements.

To analyze the effectiveness of the different methods, we use an extended use case model based on the Secure Water Treatment testbed [68].

To achieve a suitable level of protection despite of a reduced IDS quantity, their optimal placement is necessary. To address this situation, we define models for the considered attackers and the CPS under evaluation. Based on these models, we perform logical reasoning, and simulation using generic optimization, reinforced learning, brute-force methods, and heuristic breadth-first greedy search to determine the optimal placement. In summary, we make the following contributions:

- Definitions and models for CPS, attackers, and anomaly detection approaches (cf. Section 6.2.1).
- Optimization strategies considering the delay of attack detection, and node coverage (cf. Section 6.2.1.4).
- A simulation-based framework to determine the optimal IDS placement by different algorithms (cf. Section 6.2.2).
- An evaluation based on the assessment of the well-known SWaT [68] architecture (cf. Section 6.2.3).

6.2.1 Model Refinement

For a detailed analysis of the efficiency of IDS deployments, we need a thorough understanding of distributed intrusion detection systems in CPSs. Hence, we model the systems as graph structures and take note of configuration details of possible IDS in the following subsections. This model provides a common basis for the framework for IDS placement optimization in Section 6.2.2.

6.2.1.1 Cyber-physical Systems

At this point, we recall the definition of CPSs already mentioned in Section 2.2 and extend it to allow for better modeling the individual components of a CPS. While we use the same definition as before, we have a deeper look at the relations between different classes of CPS components.

We define cyber-physical systems as compound systems consisting of at least one element of each of the following three classes.

The **cyber class** C represents all elements of classical IT infrastructure. These are computing devices H, networks N, functions F, and data D.

$$C = H \cup N \cup F \cup D$$

The **physical class** P consists of involved materials M, physical process properties Z, and the physical environment E.

$$P = M \cup Z \cup E$$

Elements of C and P interact via **interfaces** S, consisting of sensors I and actuators A.

$$S = I \cup A$$

We define a cyber-physical system *cps* now as an n-tuple $cps = (i_1, i_2, \ldots i_n)$ which contains at least one element of each of the classes C, P, S. A given system tuple t is, thus, a CPS if

$$(\exists c \in C : c \in t) \wedge (\exists p \in P : p \in t) \wedge (\exists s \in S : s \in t)$$

By this definition, a CPS is a mixture of both worlds characterized by interactions and influences along its interfaces. These interactions form feedback loops, as shown in Figure 6.15. The IT systems (class C, blue) use actuators (class S, green) to influence the physical world (class P, red). Then, the measurement of sensors (class S, green) builds the basis for the decisions in the IT systems (class C, blue).

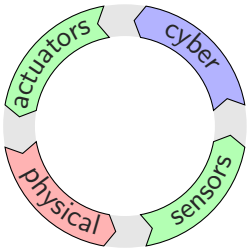

Figure 6.15: Feedback loops in CPSs.

Thus, these feedback loops represent the fundamental behavior of the CPS.

Expanding on the previous definition, Figure 6.16 shows the *classes* and *relations* between them and arranges them in a CPS meta-model. For this meta-model, we define a Function (F) as an operation that consumes data and produces other data. The function, which can be a sophisticated app, but also a simple media converter is hosted on a device H. A device is defined as a (physical or virtual) instance that can host functions and can be connected by networks N. The class data D encompasses all types of information provided by sensors I, transmitted via networks and generated or consumed by functions. An actuator A can be steered by a function to manipulate

materials M, thus, influencing their common environment E. Properties of material are measured by sensors I, which also monitor the environment. Finally, physical process properties Z describe the interactions between materials and with the environment as those influence each other.

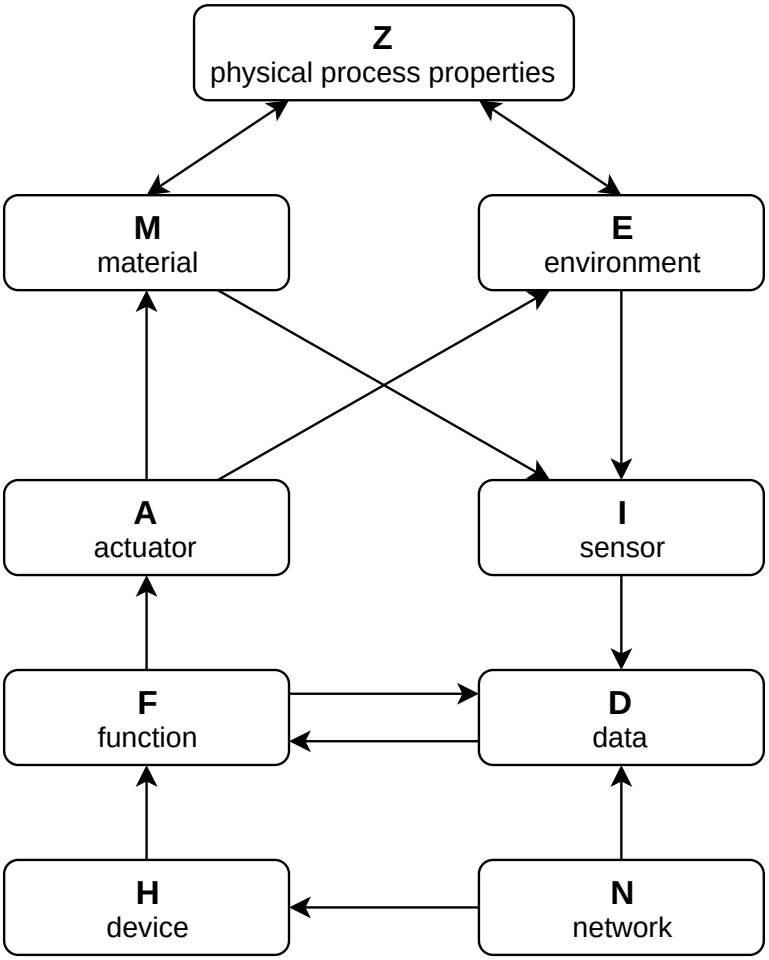

Figure 6.16: Abstract model of a cyber-physical system.

6.2.1.2 Attacker Model

Considering the previously mentioned attacks, we must assume that the main goal of an attacker is to manipulate the behavior of the CPS. Whichever component he is going to attack, he must manipulate at least one inherent feedback loop of the CPS as it defines its behavior. Being a feedback loop, the exact location of the attack is unimportant. Through the feedback, an attack always affects all components of the loop in the end.

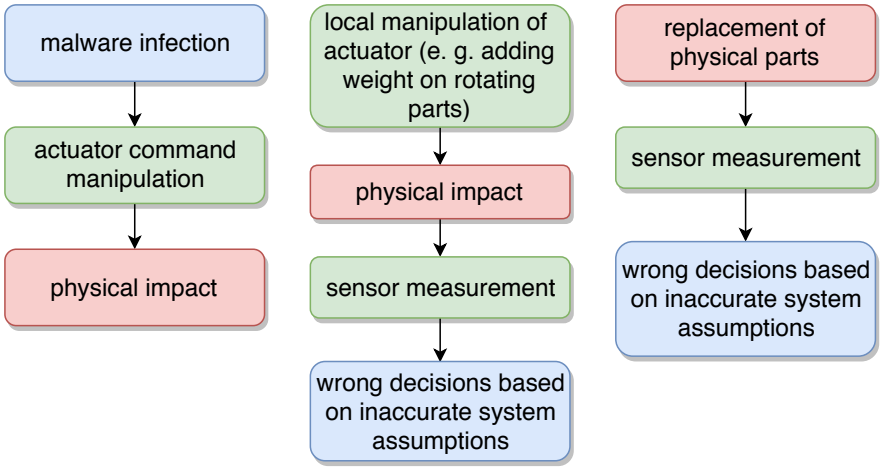

Figure 6.17: Possible manipulations in CPS feedback loops.

An attack, such as a malware infection, targeting classical IT devices may lead to manipulated actuator commands, finally resulting in physical damage (cf. Figure 6.17, left). An example of this attack sequence is the `W32.Stuxnet` attack. Brunner et al. [5] describe an initial infection by removable drives, network shares and databases, and service vulnerabilities. The infected computers, then, sent malicious control commands to the PLCs effectively manipulating the actuators.

A similar effect is possible by local manipulations of an actuator or injection of messages on the network. Measurement of the physical impact by the sensor, then, leads to wrong decisions in the controlling cyber components (cf. Figure 6.17, middle).

Alternatively, an attacker may tamper with the physical environment—for example, by replacing the material a machine is working with or altering environment conditions. This may lead to sensors operating outside their recommended operating conditions. Their measurements then again yield inadequate decisions in controlling components (cf. Figure 6.17, right).

With this model, we can classify the four different attack sequences outlined before using their first attack step:

Physical attacks on the materials or the environment directly involved in the process, first, manifest in the elements of the classes M, E.

Sensor manipulations can either happen by manipulation of the device itself, e.g., by a manipulated firmware. That is a manipulation of elements of the class I. Alternatively, these attacks could also be carried out by manipulating the network

or Fieldbus traffic from or to the sensor. Hence, a manipulation of elements of the class N is also possible [50].

Actuator manipulations can, similarly to the previous type, happen by manipulation of network traffic, i.e., elements of the class N [50]. Additionally, in CPS, manipulation of the actuator firmware is also possible in the elements of class A [124].

PLC logic manipulation can be achieved by manipulating other systems of class H on the same network to gain access to usually insecure PLC interfaces or by the network itself N. Once access is available, an attacker may dump, modify, and reprogram the PLC's logic [48, 49]. Stuxnet [3, 5] is a popular example of this type of attack.

While an *internal* attacker may execute any of these four different attack classes, an *external* attacker usually can only achieve PLC logic, sensor, and actuator manipulations from the network.

6.2.1.3 Anomaly Detection

Given these considerations, an anomaly detection system must check the integrity of all feedback loops in the CPS. Despite the formulation of appropriate rules, equations, and invariants for each loop, we need to get data, i.e., inputs for our system to detect all these attack vectors. The addition of new classes of devices, sensors, actuators, and physical materials and environments, leads to plenty of information to process.

After that and by linking and correlating all the inputs with each other, we can build a holistic model of the CPS. Figure 6.16 summarizes how components of the different classes interact.

Concerning the deployment of anomaly detection systems, we cannot collect data from every possible source in the network. On physical components, i.e., those in class $P = M \cup Z \cup E$, there is no possibility to install an IDS or a corresponding data acquisition unit (DAQ). Further, sensors, I, and actuators, A, usually do not have the resource capabilities for an IDS or a corresponding DAQ deployment. Hence in Figure 6.16, roughly only about half of the component classes can be equipped with an IDS. Without the loss of generality, we use the term IDS in the following also for the corresponding DAQ.

We represent the distribution of IDS in the overall system as a masking vector containing indicators for the presence of an IDS at any system s_j. The sum of all its elements then represents the number of IDSs present in the overall system.

$$i = \begin{pmatrix} i_0 \\ i_1 \\ \vdots \\ i_n \end{pmatrix} \tag{6.8}$$

$$i_j = \begin{cases} 1 & \text{if } s_j \text{ has an IDS} \\ 0 & \text{otherwise} \end{cases} \tag{6.9}$$

$$\|i\|_1 = \sum_j i_j \tag{6.10}$$

For every system s_j in the network we, further, define the events A_{j,t_0} and D_{j,t_0} corresponding to an attack and a detection at this system at time t_0.

We model the connections between systems as an undirected graph $G = (V, E)$ whereas V is the set of systems and E a set of edges. For every physical or logical connection between two systems, there is a corresponding edge in E. The connections reflect the relationships already shown in the abstract class model in Figure 6.16. Hence, there are edges for devices (H) on the same network (N) but also for data (D) stored on a specific device (H) and also between data (D) used and produced by functions (F).

6.2.1.4 Optimal IDS Placement

In CPS installations, the placement of intrusion detection nodes is usually restricted to a small subset. These systems include specialized equipment with power and computational constraints as well as safety certifications of devices with decade-long life-cycles. These requirements prevent any modification of them. Further, small and medium-sized economies often can not afford the installation of IDSs on every system in their network. This leads to only a small amount of IDS to cover the whole network.

To derive an optimal placement of these IDSs throughout a complex system, we define several notions of and restrictions for optimization. We distinguish between network- and host-based intrusion detection systems (NIDS and HIDS). While a HIDS, like virus scanners, may continuously check the host it is operating on, it can only detect local manipulations. Instead, a NIDS may detect all attack propagations along network links it is connected to. Thus, it may detect manipulations for several systems; however, it can only detect them once during transmission. In [73], the authors showed that an

optimal placement of NIDS in general reduces to the vertex cover problem from graph theory shown to be $NP - hard$.

Therefore, we use approximation techniques to enable optimization also for setups being too large for a theoretical solution. Additionally, literature sometimes refers to application layer intrusion detection systems. In our model, these are special versions of HIDSs with detection capabilities restricted to specific applications. Therefore, we do not consider them separately.

There are several options for optimization goals. First, we want to detect attacks as early as possible. By that, the possible impact on the overall system can be reduced due to early intervention by operators. In our model, we can measure this goal as the time difference between detection D_{i,t_1} and attack A_{j,t_0} by

$$\triangle t = \arg\min_t D_{i,t} - \arg\min_t A_{j,t} \tag{6.11}$$

Optimizing this goal, therefore, means minimizing $\triangle t$.

Second, optimizing for the highest possible node recall results in maximum coverage in the network.

$$r = \frac{|\{i \in C | D_{i,t_n} \wedge A_{i,t_m}\}|}{|\{i \in N | A_{i,t_m}\}|} \quad , t_n \geq t_m \tag{6.12}$$

For both optimization goals, we can refine them by choosing a suitable attacker model (cf. Chapter 6.2.1). While an attacker of the type *insider* may initially attack any node in the network, an *outsider* can do so only for nodes accessible from outside. We call these nodes vulnerable by outsiders as *edge* nodes and assume that they are only loosely connected to the rest of the network, i.e., they have only one network link. Usually, in CPS contexts, costs are another optimization factor. However, we include costs as a restriction for optimization rather than a goal. This allows for reducing the costs by restricting the number of IDSs n_{IDS} available while optimizing for any of the previous goals. Hence during our optimization, we always obey Equation 6.13.

$$\|i\|_1 \leq n_{IDS} \tag{6.13}$$

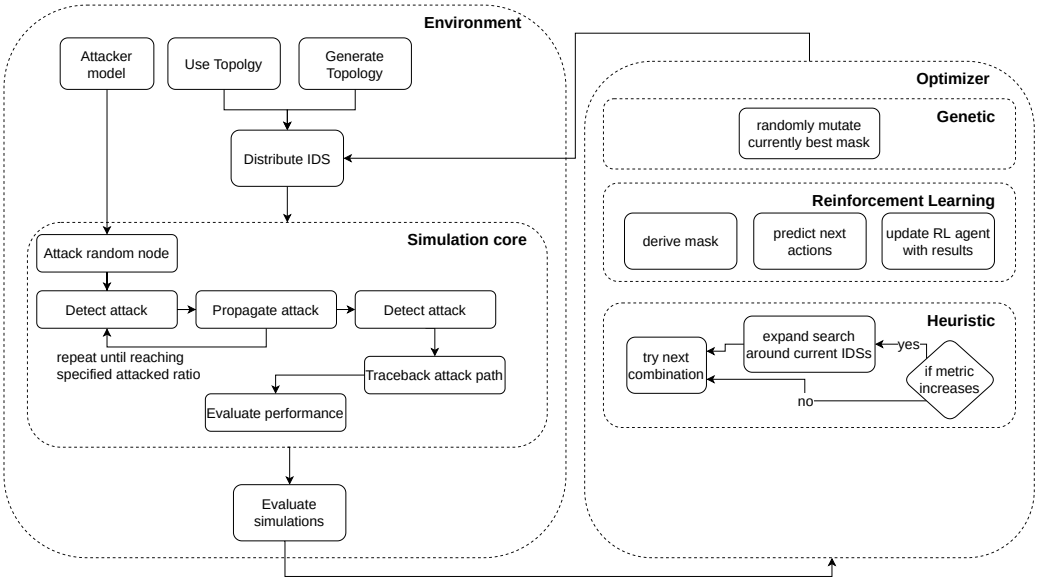

Figure 6.18: The placement estimation framework.

6.2.2 Framework for Optimal Placement

The framework consists of three basic components: a simulation core, an environment, and an optimizer.

The environment provides the attacker model and a topology. This information is used to place n_{IDS} IDSs throughout the network initially. Then, it initializes the simulation with a random attack on one node an attacker of the current model has access to.

Using the simulation core already presented in more detail in Section 6.1.1, we then evaluate the propagation and detection throughout the network. Finally, the simulation core provides the metrics described in Section 6.2.1.4.

A controller runs the simulation for several trials and averages the results to minimize the impact of the random initial attack choice.

After that, one of the different optimizers modifies the current IDS distribution and restarts the simulation.

Figure 6.18 summarizes the overall framework and shows how the different components interact. In the following, we describe the components in more detail.

6.2.2.1 Simulation Core

The simulation of a specific IDS placement uses several rounds, i.e., timesteps, to derive the described metrics. Throughout the simulation for any system s_j the event D_{j,t_0} represents a detection with the IDS of s_j at time t_0. Correspondingly, A_{j,t_0} corresponds to an attack of that system.

For each timestep t, every system s_j which has been attacked in the previous timestep t_0 uses its IDS to detect the attack with a specified detection rate

$$d_j = P(D_{j,t}|A_{j,t_0}) \tag{6.14}$$

Additionally, every system s_j which has not yet been attacked but has an IDS may rise a false alarm with a specified false alarm rate

$$f_j = P(D_{j,t}|\neg A_{j,t_0}) \tag{6.15}$$

After that, for every neighbor s_i of an attacked node s_j, the attacked node propagates its attacked state with the attack propagation probability i_j

$$i_j = P(A_{i,t}|A_{j,t_0}) \tag{6.16}$$

The first two parameters, therefore, describe the capabilities of the considered IDS. For our analysis, we assume that these parameters are the same for all used IDSs. The model itself does not depend on them being static or global. Hence, every IDS can have different and also changing detection and false alarm rates. This allows for modeling online learning detection methods.

The third parameter, the attack propagation probability, concerns the attacker model. With a high propagation probability an attack spreads quickly through a system network leaving only little time to detect it. However, some attacks require user interaction in their spreading or exploit process, e.g. by clicking a link or opening a file, or actively defer their exploits to later points in time. These can be assigned with a lower propagation probability.

Still, over time all attacks have the potential to spread throughout the whole network.

For every two nodes s_i, s_j, which have an IDS and detected an attack, we consider all other nodes on the shortest path from s_i to s_j as reporting an *assumed detection*. This

enables us to report detections even for nodes without a deployed IDS. An attacker might not necessarily choose the shortest path to reach his attack goal. However, we could include exploitation difficulty or probability for every link between nodes as their distance. Then, the shortest path between two reported detections is also the most likely one taken by an attacker. While our shortest path reasoning approach only serves as an example, the overall framework for placement optimization applies to other reasoning or heuristic methods as well.

During the simulation, once a specified ratio of nodes in the network is in its attacked state or a specified number of time steps is reached, we execute a backtracking algorithm to estimate which nodes got attacked but cannot detect their state due to a missing IDS. For that, for every new detection we calculate the shortest path to any previous detection in the whole network and label all intermediate nodes as *assumed detection*s. This heuristic serves to enable a restricted IDS distribution to cover larger networks. With all that information, we then calculate the recall of found detections as well as the time needed for the first detection to occur after the initial attack. We only label detections with an attack in a previous time step as a *true positive*, all others as *false positive*. Not detected attacks are labeled as *false negative*, while no detection and no attack yield a *true negative*. Hence we can calculate the recall in the overall network with

$$rec = \frac{tp}{tp + fn} \tag{6.17}$$

As we use Monte Carlo simulations for our method, we average the results over several executions of the same setup.

6.2.2.2 Optimal Placements

We can now derive optimal solutions for specific system setups with the previously outlined model and optimization goals.

For the first optimization goal, the shortest time to the first detection, let us first assume that only one node has an IDS. The time it takes to detect the attack equals at least the shortest distance from an attacker's entry point to the node with the IDS. The time cannot be shorter as there is no shorter path for the attack to spread to the IDS. However, it can be larger if the attack did not spread along the shortest path. As we use this theoretic optimum as a baseline for later approximations, a lower bound on the time to the first detection is sufficient. If we now consider several nodes with IDSs, the lower bound equals the shortest path from the entry point to the shortest path of

any node with an IDS. Thus, we are looking for a set $\mathcal{D}^*$ of IDSs nodes which gives us the lowest sum of $\triangle t$ to any possible entry point of attacks. We denote the set of possible attack nodes as $\mathcal{A}^*$.

$$\min(\triangle t) = 1 + \sum_{A_i \in \mathcal{A}^*} \min(dist(D_j, A_i) | \forall D_j \in \mathcal{D}^*) \tag{6.18}$$

To determine this lower bound in our model, we use a modified version of the Dijkstra algorithm to calculate the shortest path between one node and a set of nodes. The best distribution of any number of IDSs must then yield the lowest sum of all these shortest distances. Algorithm 4 outlines how this search can be done. For each possible combination of IDSs, it calculates the distances from one IDS node to all other nodes (lines 3–8). The sum of the distances from every possible entry point to their closest IDS corresponds to the minimum detection time with that specific setup (lines 9–17). Finally, we can then identify the setup with the lowest sum as the one with the fastest detection capability (lines 18–21).

For the second goal, the biggest coverage of nodes in the network, we need to recall the model of assumed detections. As we label all nodes between two IDS nodes with detections also as *assumed detections*, we reach biggest coverage, when these shortest paths are long and do not overlap. Algorithm 5 outlines this search. For each pair of IDS nodes, we calculate the shortest path between them (lines 3–6) and add all the nodes on this path to a set (lines 7–12). Finally, the combination of IDS with the largest covered set is the one with the biggest coverage in the network (lines 13–18).

6.2.2.3 Approximating the Optimum

The presented algorithms for both detection goals have a severe drawback. They are slow for larger networks of systems. As they both involve testing every combination of n_{IDS} out of N systems and running the Dijkstra algorithm several times, they do not scale. Hence, we also established three different optimization strategies that try to lessen these computation costs.

Genetic Optimization In general, evolutionary algorithms first rank several possible solutions, also called population, of the problem according to a metric. A specified portion of the best solution candidates is then used for the next round of optimization. Randomly recombining some of these best solutions creates new possible solution can-

Algorithm 4: Determining combination with fastest detection.

```
   Input: Number of IDS
   Output: List of best nodes
 1 minComb = none;
 2 minSum = inf;
 3 foreach combinationWithNNodes do
 4 |   distSum = 0;
 5 |   dists = [];
 6 |   for n in combination do
 7 |   |   dists.append(dijkstra(n));
 8 |   end
 9 |   foreach entryPoint do
10 |   |   minDist = inf;
11 |   |   for d in dists do
12 |   |   |   if d[entryPoint] ¡ minDist then
13 |   |   |   |   minDist = d[entryPoint];
14 |   |   |   end
15 |   |   end
16 |   |   distSum += minDist;
17 |   end
18 |   if distSum ¡ minSum then
19 |   |   minComb = combination;
20 |   |   minSum = distSum;
21 |   end
22 end
23 return minComb;
```

didates. With every round, we then assume that the better solutions evolve just like in nature's evolution.

In our simulation, we define a population as one distribution vector i (cf. Equation 6.8). The optimizer starts with a random mask respecting the simulation restrictions, i.e. no IDSs on nodes of classes in P or S. We use several runs of the simulation with every population to eliminate the effects of the random entry point choice during the simulation. Then, we use the average of the considered optimization goal and determine the so far best distribution vector. We, then, randomly move n IDSs and repeat this evolution for several generations. By executing multiple populations in parallel, we reduce the likelihood of getting stuck in local optima. Note, that we, finally, return the best-encountered population and not necessarily the last one. As our fitness

Algorithm 5: Determining combination with the biggest coverage.

```
   Input: Number of IDS
   Output: List of best nodes
 1 maxComb = none;
 2 maxSum = 0;
 3 foreach combinationWithNNodes do
 4 |   nodeSet = set();
 5 |   foreach combinationOf2 do
 6 |   |   distances, predecessors = dijkstra(pair[0]);
 7 |   |   cur = predecessors[pair[1]];
 8 |   |   while cur ≠ predecessors[pair[0]] do
 9 |   |   |   nodeSet.add(cur);
10 |   |   |   cur = predecessors[cur];
11 |   |   end
12 |   end
13 |   if length(nodeSet) ¿ maxSum then
14 |   |   maxSum = length(nodeSet);
15 |   |   maxComb = comb;
16 |   end
17 end
18 return maxComb;
```

calculation for the population already equals our optimization goal, we can directly output the fittest one.

Compared to the evaluation of the real optima described before, this computation does not depend on the network's size . However, it is yet unclear how many generations we need for meaningful results.

Reinforcement Learning To apply a reinforcement-learning based optimization, we model the procedure of distributing IDSs as a game.

In current reinforcement-learning strategies, an agent is trained to choose an action based on a current state. At some time, an action may lead to generating a reward for the agent. This reward is then used in a back-propagation optimization method to train the agent's underlying neural network [125].

For our optimization, the learning agent can either *place* an IDS at its current location, choose the *next* network link, or *walk* along the current network link. Thereby, its current state is represented by the so far chosen IDSs, the current position of the agent, and the current network link. The action *next* changes the *next link* state to the next node from the adjacency list of the *current node*. At the end of the list, we simply

repeat it from the beginning. With the *walk* action, the current node changes to the one pointed by the *next link* state. Also, the *next link* state changes to the first entry of the adjacency list of the new *current node*. The *place* action, sets an IDS at the *current node*. The agent cannot undo this decision within the same game. If placing an IDS results in the maximum number of IDSs reached, the game is over and rewards are generated. For this, the simulation runs using the provided IDS distribution vector and outputs the current metric as the reward for the reinforcement agent. Thus, the reward for the last placement action equals the value of the chosen optimization goal. In the case of optimization for time to the first detection, the reward for the last placement action r_n is the sum of shortest distances from any entry point to any from the agent chosen IDS node. For all previous actions, rewards are calculated by an exponential decay. The reward r_t at time t is

$$r_t = r_n \cdot \gamma^{n-t} \quad \forall t < n \tag{6.19}$$

If we let our learning agent play this game several times, we assume that it gets better over time in choosing the right positions for IDSs. Similar to the previous method, it is unclear, how many rounds we need to play. Figure 6.19 depicts the structure of the used reinforcement agent for the optimization. The agent consists of a relu and a softmax layer to decide for an action based on the input state.

To train the agent, we run successively 2000 games with our agent. At the beginning of each game, we place the agent at the same node in the network, i.e., the game's initial state is always the same. After this, the agent takes turns by choosing a random action whereas the probability of each action equals the output of the softmax layer with the current state.

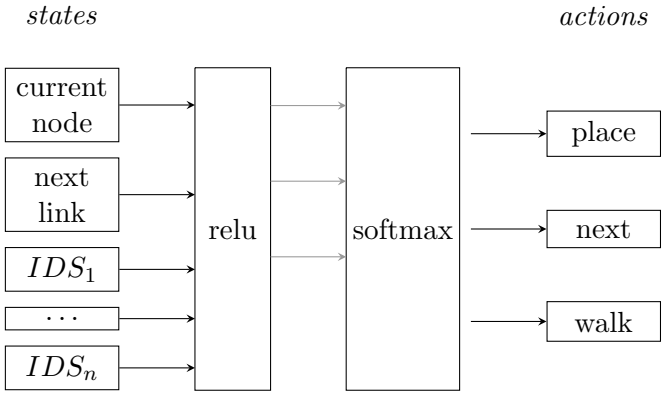

Figure 6.19: Example of a learning agent for optimization of two IDSs with the described states and actions.

Heuristic Breadth-first Greedy Search Finally, we use customized variants of the previously described optimal methods. In a heuristical approach, we start with a random distribution vector respecting the simulation restrictions. If the current optimization goal is enhanced by a distribution vector, we expand a breadth-search at each currently chosen IDS for one more level. We add all new combinations to a list of the current search and proceed with the next combination in it.

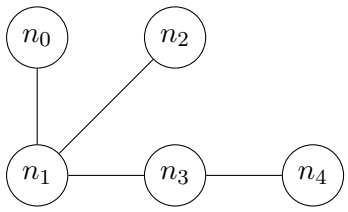

Figure 6.20: Example of heuristic breadth-first greedy search.

In the example in Figure 6.20, assume (n_0, n_1) is currently the best solution for the current optimization goal. Then the heuristic search will explore all the following combinations $(n_0, n_2), (n_0, n_3)$. In the next step, we will only expand to n_4, if (n_0, n_3) yields a better result then the previous (n_0, n_1).

We use this approach with the assumption that the optimization goal is enhanced while virtually approaching the optimal placement. For the time to the first detection, a move in the direction of typical attack entry points should yield a decrease in its value. Similarly, a move towards nodes providing a large number of connections to other nodes, e.g., a router, should increase the overall coverage.

6.2.3 Experimental Results

As an example system, we use the first stage of the Secure Water Treatment (SWaT) architecture presented by [68]. Figure 6.21 shows part of the testbed modeled according to our abstract CPS model. Every node is represented by one of the boxes with its corresponding node class as a bold letter. The solid lines indicate connections between nodes in the same layer, while connections to nodes in different layers are represented with red dashed lines. The bottom and middle layers contain the components of the first stage as presented in [68]. However, we added the top layer providing example use cases for the architecture. We assume that the sensors in the architecture deliver sensor data, which is stored in a historian and used for the process control. In addition, the pumps themselves provide pump data referring to their current status. These are also part of the data storage but may be used for performance evaluations of the pumps.

This performance evaluation also uses the control commands generated by the process control.

While the presence of data storage and process control functions is inspired by the SWaT architecture itself—there is a historian and a programmable logic controller—we added the pump performance evaluation to include a predictive maintenance scenario.

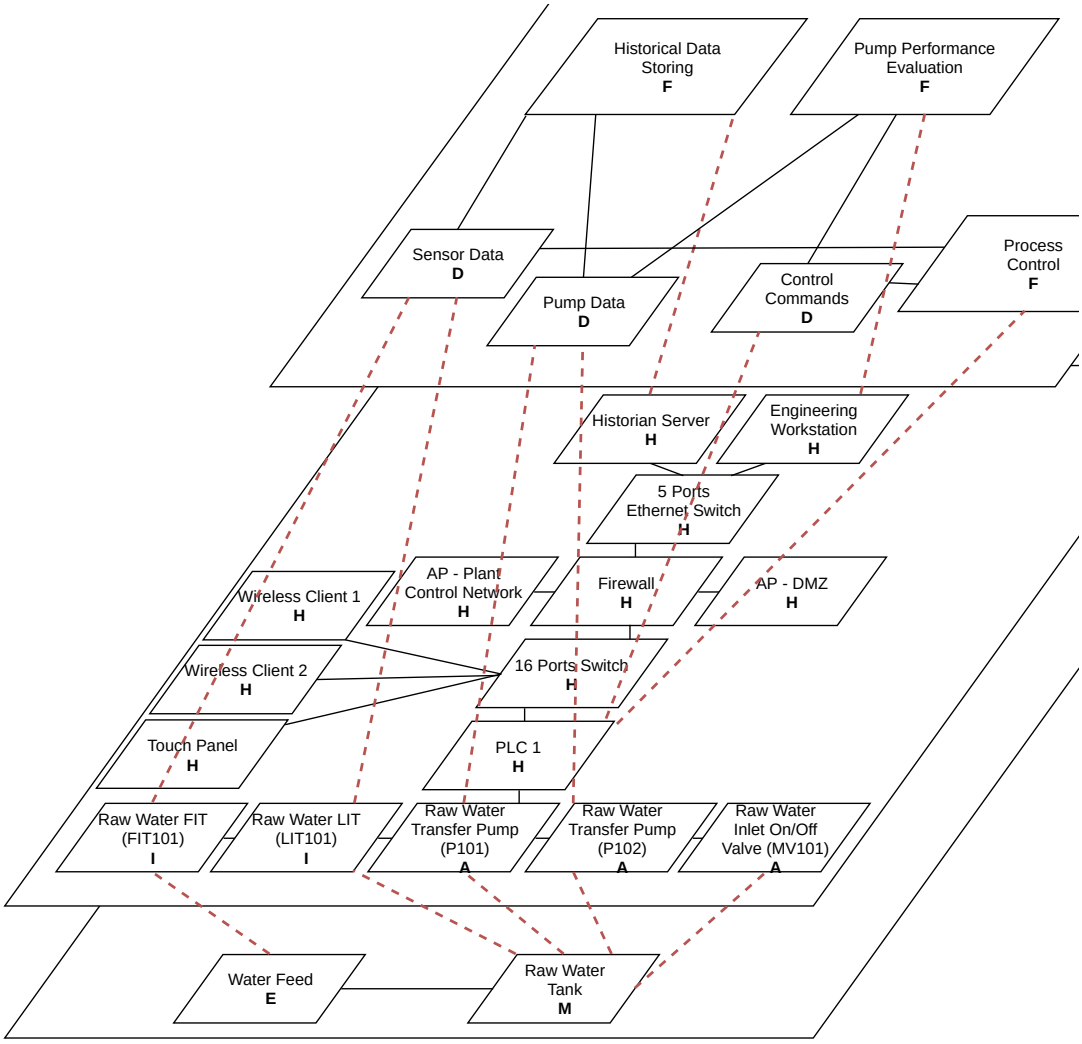

Figure 6.21: Stage P1 of the SWaT architecture with extended use case scenarios modeled with the provided CPS abstract model. The class of each component is provided in bold print.

For our experiments, we restricted all sensors, actuators, as well as the physical components not to have an IDS. All other nodes in Figure 6.21 may, thus, deploy an IDS. During the simulation, we assume that all IDS have the same detection and

	insider			outsider		
	2	3	4	2	3	4
Optimal	2.38	2.17	2.00	2.00	1.60	1.40
Random	3.07	2.71	2.47	3.20	2.80	2.51
Worst	4.29	3.83	3.47	5.40	5.00	5.00

Table 6.2: Baseline for time to first detection performances.

false alarm rates of 95% and 5%. While the framework allows specific values for every IDS, this assumption eases the later interpretation and verification of the results. The attacks propagate with a probability of 70% and a simulation run stops after 20 time steps. The metrics are derived as the average of 200 runs of each configuration.

For a more fine-grained analysis, we distinguish two types of attacker models, i.e., insiders and outsiders. While insider attacks may initially occur on any node in the network, outsider attacks are restricted to nodes with only one network link for the very first attack step. In Figure 6.21, these are the following nodes: wireless clients 1 and 2, the touch panel, and the access points placed in the Plant Control Network (PCN) and the Demilitarized Zone (DMZ).

Overall, for every optimization goal and method, we see a performance increase with more IDSs. While we expect that behavior, it serves as a baseline test of the underlying simulation.

6.2.3.1 Time to First Detection

In Tables 6.2–6.3, we show the times needed until first detection of an attack in different scenarios. For each of the two attacker models, *insider* and *outsider*, we evaluate the performance of 2, 3, and 4 IDS in the network.

As ground truth, we derived the best, worst, and average results over all allowed IDS placements in the whole system. For this, we evaluate for each valid IDS placement the shortest path between any node without an IDS to one with an IDS as described in Chapter 6.2.2.2. This path's length, then, relates to the time needed for an attack from that node to spread until the first IDS may detect it. In the *outsider* attacker model, we only evaluate the paths from one of the edge nodes. Finally, we need to add one time step to the determined path length to account for the detection time (cf. Equation 6.14). The best and the worst results we obtain with this method for a specific IDS placement represent the optimal and worst solution in this setup. The

	insider			outsider		
	2	3	4	2	3	4
Genetic	**2.4**	**2.25**	**2.10**	**2.35**	**2.1**	**1.57**
RL	2.49	2.28	2.16	2.41	2.1	2.00
Heuristic	2.77	2.57	2.16	3.01	2.38	2.32
Improvement	21.8%	17.0%	15.0%	26.6%	31.4%	39.0%

Table 6.3: Performance of different optimization methods for optimized time to the first detection. Best performances for every setup in bold.

average over all placements is what we would reach without any optimization but by choosing a placement randomly.

Table 6.2 shows that an increasing number of IDSs yields shorter times to the first detection. Comparing the times to first detection, $\triangle t$, under the two attacker models, we notice that all methods yield shorter detection times for *outsider* attacks than for *insider*s. Looking at the random placements, we see that there is no significant difference in the detection time between different attacker models. However, worst placements are more dramatic if possible entry points for an attack are restricted to a limited set, i.e. in the *outsider* attacker model. On the other hand, it is possible to achieve better optimal results in the *outsider* model than in the *insider* model. That means once we know where attacks are possibly coming from, we can optimize the positioning of our IDSs accordingly. Since the set of nodes where an attacker may first touch any of the available systems is limited in the *outsider* attacker model, placing an IDS near this limited set of systems significantly decreases the time to the first detection. However, as we define a minimum detection time of one time step in our system model (cf. Equation 6.14), we cannot have faster detections than one time step.

For the optimization methods, *Genetic*, *RL* (reinforcement learning), and Heuristic, still more IDSs yield shorter detection times. Overall, the *Genetic* method yields the best results.

To compare our approximated placements to the optimal baseline, we use the *improvement* over the random choice. Overall, we reach improvements from 15% to 39% (cf. Table 6.3). However, the improvements show different characteristics for the two attacker types. For the *insider* attacks, an increase in IDSs results in less improvement. As in this attacker model there are fewer restrictions on the entry point, the best position must support the detection of attacks from every other node equally. Hence, with an increasing amount of IDSs in the network a random placement converges to

	insider			outsider		
	2	3	4	2	3	4
Optimal	0.25	0.46	0.58	0.25	0.46	0.58
Random	0.16	0.26	0.35	0.16	0.26	0.35
Worst	0.08	0.13	0.17	0.08	0.13	0.17

Table 6.4: Baseline for optimized node coverage performances.

the optimal placement. Therefore, our relative improvement reduces the more IDSs we use.

On the other hand, for the *outsider* attacks, an increase in IDSs also raises the achievable improvement as we approach placing an IDS directly at every possible entry point.

6.2.3.2 Node Coverage

In the node coverage test, we place n_{IDS} IDSs throughout the network. As an example for distributed anomaly detection with limited information gathering, we use the simple heuristic approach described at the end of Chapter 6.2.2.1. Table 6.4 shows the achieved ratio of all nodes in the system which can be covered with a specific setup.

We can calculate the optimal and worst case positions only easily when every system is already in its attacked state. Then, however, the actual node coverage in the network is independent of the attacker model as the source of this attacked state is not relevant to the metric. Therefore, there is no difference in the theoretical best and worst case placements.

Again, an increase of n_{IDS} leads to higher node coverage. We reach the worst case concerning the node coverage for every scenario when every deployed IDS only detects attacks locally, as shown in Equation 6.20.

$$wc = \frac{n_{IDS}}{|S|} \tag{6.20}$$

We derive the optimal placement as described in Chapter 6.2.2.2. Intuitively, we obtain this best case by maximizing the sum of the distances between any two deployed IDSs. However, we actually need to maximize the number of distinct nodes on the shortest path between any two deployed IDSs. The random placement equals the average of node coverage of all possible IDS placements.

	insider			outsider		
	2	3	4	2	3	4
Genetic	**0.24**	**0.32**	**0.36**	**0.24**	**0.33**	0.37
RL	0.23	0.28	0.31	0.23	0.28	0.33
Heuristic	0.23	0.28	0.35	0.23	0.30	**0.40**
Improvement	50.0%	23.1%	14.3%	50.0%	26.9%	14.3%

Table 6.5: Performance of different optimization methods for optimized node coverage. Best performances for every setup in bold.

Comparing the different approximation methods, again reveals the *Genetic* method as best performing. However, in this case the difference to the other methods is less pronounced as they obtain almost identical results.

Comparing the optimal and worst placements to our approximated solutions, we can reach improvements from 14% to 50% (cf. Table 6.5). While the best improvement can be seen with only two IDSs, it decreases the more IDSs we use. The optimal solution distributes the IDSs evenly over the network. In a fully connected network, any combination would work the same. However, as most real-world networks are not fully connected, an increase of n_{IDS} only leads towards that even distribution and, hence, the average placement converges to the optimal.

Overall, the approximated placements almost yield the same performance as an average placement with one additional IDS. Hence, by applying our method, we can either work with one IDS less while almost maintaining the performance or extend our performance to a level we cannot afford or achieve.

6.2.4 Conclusion

Within the previous sections, we present a comprehensive model for CPSs as well as two different attacker schemes of interest for the Industry 4.0. Based on this model, we develop assessment criteria for the performance of a distinct IDS placement in a larger network. Using our simulation framework from Section 6.2.2.1, we then evaluate that performance for different IDS placements. With three different methods, including heuristics, reinforcement learning, and evolutionary algorithms, we are able to optimize the placement of IDSs. In the extended use case example of the Secure Water Treatment testbed, we analyze the placement according to the previously defined criteria. Thereby, we achieve an optimization of the time to detection of up to 39% and an increase of node coverage of up to 50%.

While we use full exploration of all possible placements as a baseline for verification of our results, the optimization methods each only evaluate a small portion of the possible deployments. Where a full exploration of all possibilities is infeasible for larger setups of CPSs, our optimization methods can still provide approximations to the best solution.

For this, we need a network model of the CPS as described in Section 6.2.2.1. Without doubt, the creation of such a model requires some effort of a CPS operator. In turn, however, our suggested methods deliver optimal IDS positions for smaller systems (cf. Section 6.2.2.2) or at least good approximations to these for all other systems (cf. Section 6.2.2.3). In reality, it will always be the case that the amount of IDS sensor nodes is restricted by budgets and resource constraints of the devices to monitor. Knowing how to allocate the available resources wisely may boost the detection performance.

Though we base our work on simulations and abstract assumptions on IDSs, we still assist CPS operators in making design decisions for future IDS deployments. Using simulations allows for analyzing different setups very early in the design phase of a CPS. Whereas security architects previously made best-guess decisions of how many IDS sensors to use and where to place them, we provide a method of quantifying the benefits of particular placements. Applying one of the three optimization methods even allows for deriving better placement decisions. Overall, this shall ease the usage of intrusion detection systems in large and complex network scenarios.

6.3 Understanding Advanced Attack Procedures in CPSs from Heterogeneous Logs

A large group of researchers and CPS operators still criticize the use of machine learning methods in the context of anomaly detection as being inscrutable black-boxes. Redmiles [111] suggests that a lack of transparency in alert notifications reduces their efficacy.

In addition, different studies as well as industry practitioners report difficulties with implementing IDSs. A large amount of cumbersome manual work is required, especially in setting up rules for detecting actual threats in available data. For this, we provide a semi-automated method for managing different data sources and evaluating them without deeper IT security knowledge.

Thereby, we focus on two highly important goals for system operators:

- Closing the initial entry point of any attack ensures that attackers may not reuse it for further attack attempts. Any efforts for regaining control over systems and removing compromised or manipulated parts are pointless if the attacker may just redo his steps to reach the same state again. Hence, methods automatically narrowing down the probable entry points are of great help for operators and analysts in charge.
- Additionally, after closing the entry point, the propagation throughout the own network reveals which possibilities or even vulnerabilities attackers used to reach restricted systems. Therefore, further tracking down the progression of attacks allows for identification and later fixation of systematic problems in the own network.

6.3.1 Concept

In the following subsections, we explain the different steps of our log aggregation, classification, and understanding model. After identifying suitable logs and collecting their information on a central point (Section 6.3.1.1), we automatically estimate the relevance of every log file and their entries (Section 6.3.1.2). With additional input of a system operator, these ratings may be enhanced for deployed intrusion detection systems. This information then allows for identifying attacks (Section 6.3.1.3) as well as understanding their progression throughout a company's network (Section 6.3.1.4).

System Type	#log files	est. running times
default archlinux install	1	1 day
default debian install	25	days
NAS server	16909	years
development machine	44221	years

Table 6.6: Number of files named "*.log" on different system types.

6.3.1.1 Log Aggregation

On ordinary workstations, data servers, and embedded devices, operators usually have access to a vast number of log files. Often, these serve for debugging purposes.

During advanced cyber attacks, attackers use vulnerabilities in several software artifacts. While different installations come with different logs per default, the number of log files present increases over time and depending on the software used. Table 6.6 lists examples for some systems and usage times.

All these log files grow continuously collecting information especially on unusual system behaviors. In addition, since they contain important information for the identification of errors, they also constitute a valuable source for attack detection and forensic analyses. Hence, to use this data during attack detection, we need to make it accessible for analysis systems.

Depending on the resource capabilities available for a machine, we distinguish two types of log and information aggregation:

Constrained systems allow a monitoring node to connect directly. This monitoring node has direct access to log files. Additionally, the constrained system may provide small status endpoints that return status information on request. Hence, the monitoring node *pulls* available information from the constrained system. This system type also corresponds to proprietary systems that have some built-in logging or status reporting capabilities but do not allow the installation of additional software.

Powerful systems forward every new log entry directly to an offsite analysis system. System operators may choose to deploy specific information aggregation procedures enriching the information with data not written to logs. Hence, the system itself *pushes* available information to the analysis system.

The monitoring system itself can gather information outside the scope of a single machine. These are lists of connected devices and their network interaction statistics. This setup results in centralizing the de-central information sources, as shown in Figure 6.22.

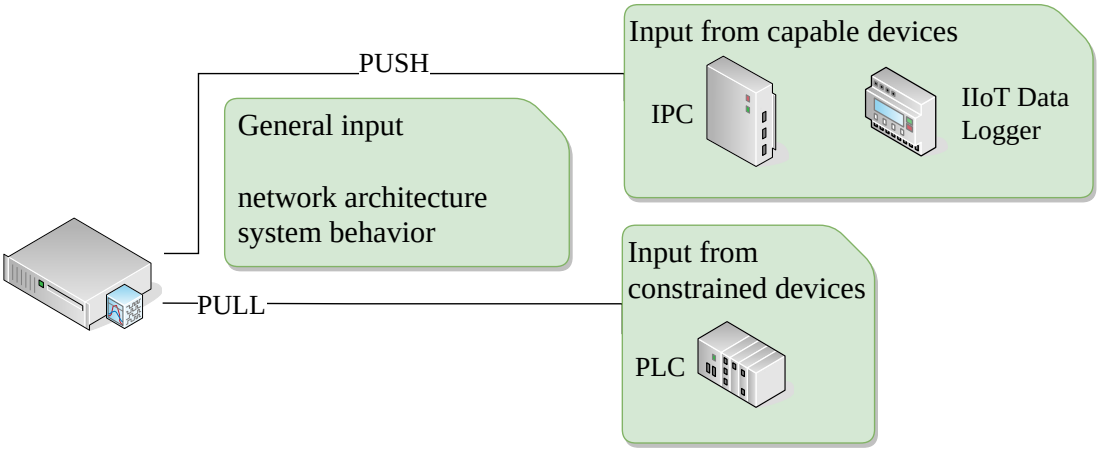

Figure 6.22: Data acquisition architecture.

6.3.1.2 Alert Log Classification

While the acquired log files can contain arbitrary data, they always have a timestamp associated with their entries. Hence, we denote a log file as a sequence of log entries

$$\mathcal{L} = \{(t_0, e_0), (t_1, e_1), \ldots, (t_n, e_n)\} \tag{6.21}$$

with e_i being the log message at time t_i. We denote subsets of a log file up to the time t_k as

$$\mathcal{L}_k = \{(t_0, e_0), (t_1, e_1), \ldots, (t_k, e_k)\} \tag{6.22}$$

Each log can be categorized according to its update interval, its information layer and its information content. We call log files getting regular updates, like the output of watchdogs, *PULL* logs as the regularity often originates in timer-based command execution. We denote other log files as *PUSH* logs as, for those, new entries are triggered by external events. Concerning the technological abstraction, log files contain either technical or functional information. Technical information may include available network links, open ports, or firewall logs. Functional logs contain information from the corresponding application as web server logs or production machine log output.

Each log can either contain status information or event indicators. Status updates, as we usually see in web server logs, are denoted as *component* logs. The output of an anomaly detection system, however, contains only event notifications. Hence, we call them *indicator* logs.

	Component Logs	**Indicator Logs**
functional	Plans	application error logs
	allowed/dynamic communication	
	allowed/dynamic services	
technical	Services	suricata
	open ports	fail2ban
	Web interfaces	iptables
	SW Endpoints	

Table 6.7: Examples for log file classification.

By this definition, *indicator* logs are always *PUSH* logs, since their entries are triggered by non-regularly occurring anomalous events. However, *component* logs with regular updates containing information about a system's status may be converted to *indicator* logs by extracting the changes in subsequent entries, i.e., $e_i - e_{i-1}$. Considering the *PULL* log P, we derive a corresponding *PUSH* log $\mathcal{L}^P$ as

$$\mathcal{L}^P = \{(t_i, e_i - e_{i-1}) \,|\, (t_i, e_i), (t_{i-1}, e_{i-1}) \in P : e_i \neq e_{i-1}\} \tag{6.23}$$

Table 6.7 gives examples of log files or log file generators and categorizes them using the provided terms.

Identifying status changes Current big data information aggregation systems usually use semi-structured information based on JSON [126]. We adopt this mechanism to store structured as well as unstructured logging data. While log files written to a local hard disk usually use plaintext to describe their entries, status information is often retrieved in structured formats. For example, a REST API of a service usually responds with a JSON string. Thus, we also convert the plaintext entries into a structured representation relying only on the definitions in Equation 6.21. Thus, we require a small converter for every log type to extract the corresponding timestamp.

Having a uniform format (cf. Listing 6.1), we can now detect status changes in subsequent entries in the same log. Listing 6.1 shows the same information as in Listing 5.1 after the conversion to a JSON representation. In Algorithm 6, we use a dynamic programming based approach to extract these changes. First, every log entry is converted to its structured representation (l. 4). For every key in the new entry, we record changes according to its last retrieval. To initialize this process, we do not consider the first occurrence of a key as a status change. This allows for incorporating services not responding with all status information on each request.

```
{"timestamp": "2019-02-24 18:10:30.634900",
 "message":"[**] [144:1:1] (spp_modbus): Length in Modbus MBAP header
     does not match the length needed for the given Modbus function.
     [**]
 [Classification: Generic Protocol Command Decode] [Priority: 3]
 02/24-18:10:30.634900 192.168.1.104:502 -> 192.168.1.100:4386
 TCP TTL:128 TOS:0x0 ID:38534 IpLen:20 DgmLen:57 DF
 ***A***F Seq: 0xBF6CC890  Ack: 0xCF15D076  Win: 0xFADF  TcpLen: 20"
}
```

Listing 6.1: Snort alert entry in minimal structured format.

Algorithm 6: Detecting changes in subsequent status log entries.

Input: Log $\mathcal{L}$
Output: Set C of changes

$p \leftarrow \{\}$
$C \leftarrow \{\}$
foreach $(t, e) \in \mathcal{L}$ **do**
  $j \leftarrow$ `getJsonRepresentation` (e)
  foreach $key, value \in p$ **do**
    if $key \in j \wedge j[key] \neq value$ **then**
      $C[t][key] \leftarrow (value, j[key])$
    end
  end
  foreach $key, value \in j$ **do**
    $p[key] \leftarrow value$
  end
end
return C

While these status changes may justify raising an alert, operators need to apply thresholds to distinguish between expected and unexpected changes manually.

6.3.1.3 Attack Understanding

The initial point for our attack understanding process is an alert found in an indicator log. Upon detection, we disassemble the corresponding alert message by Natural Language Processing (NLP) techniques and search for the terms in other log files in a short time window before and after the alert.

We separate the words of the message using two methods available in the python spacy library [127]. First, we look for noun chunks. These are phrases, i.e., parts of the

```
1 (spp_modbus
2 Length
3 Modbus MBAP header
4 the length
5 the given Modbus function
```

Listing 6.2: Noun chunks in the snort alert message from Listing 6.1.

```
1 spp_modbus
2 Length
3 Modbus
4 MBAP
5 header
6 length
7 Modbus
8 function
```

Listing 6.3: (Proper) nouns in the snort alert message from Listing 6.1.

message, containing a noun as well as describing adjectives. Then, we search through the other log files in the specified time window for the extracted phrases. If we find one of these phrases, we assume a high relevance of the found log entry. Since noun chunks usually consist of several words, it is rather unlikely to find the exact same sequence of words in other logs if they are not related. Listing 6.2 shows the chunks derived from the example in the previous section.

Additionally, we apply part-of-speech tagging to the alert message to identify nouns and proper nouns. Searching for these in other log files may result in log entries having information correlated to the reported event. As this may include unspecific terms like `header` or `length`, we assume a lower relevance rating for the found log entries (cf. Listing 6.3).

With this procedure, we automatically enrich every upcoming alert of deployed intrusion detection systems with possibly interesting information from other log files. This enables operators to easier understand possible causes of the incident and additional information on the possibly affected systems. We can further enhance this method by including specific lookups. For example, we can extract IP addresses from the alert message using a regular expression (regex). Also, a regex for URLs or domain names in the alert may yield further interesting information.

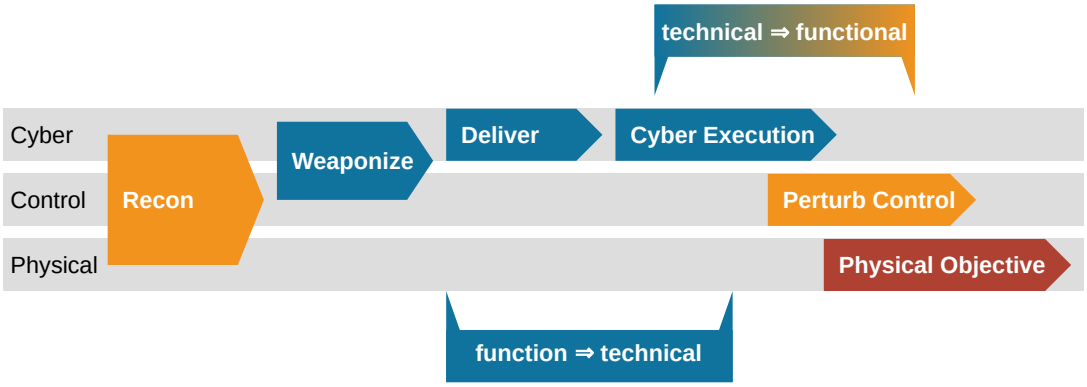

Figure 6.23: Mapping of log entry transitions into the cyber kill-chain for cyber-physical systems by [114].

Finally, this gives us a list of log entries throughout several log files, maybe even from different systems, corresponding to one alert of an IDS.

6.3.1.4 Attack Tracing

Using the found log entries in the classified alert logs, tracing attacks throughout multiple systems in a network is possible. For this we use the concept of cyber kill-chains as described in [128]. In the cyber kill-chain different activities an attacker might execute are aligned on a timeline to visualize their order and dependence on each other.

We start with alerts found in indicator log files. They report anomalies for different systems and contain valuable information in their alerts. By separating individual information in these alert entries and searching for them in component logs (cf. Section 6.3.1.3), we discover earlier related information of an attack in other common log files. These entries may be caused due to compromised user accounts, leaked credentials, or insecure configuration of subsystems.

Let a_n be an alert at time n and e_m a related log entry at time $m < n$. We then call the tuple $t_{m,n} = (e_m, a_n)$ a transition. Depending on the type classes of e and a, we, thus, see two different types of transitions. For this, we adopt the cyber kill-chain for CPSs as proposed by [114]. In general, it follows the well-known definition of attack steps like: *reconnaissance* for information retrieval, *weaponization* for development and acquisition of attack tools, *delivery* for the transfer of attack code to a target, and *execution* for the finally targeted attack. For CPS, Hahn et al. [114] divide the execution step on the three different layers (cyber, control, and physical) with the assumption that the final objective of an attacker is a specific manipulation of the physical process.

data type	method	type
executed software	logstash command output	*pull*
application data	logstash command output	*pull*
log data	logstash log output	*push*
local services	logstash command output	*pull*
remote services	metricbeat HTTP query	*pull*
system behavior	metricbeat system info	*pull*

Table 6.8: Implemented types of information sources in our demonstrator.

Also, they map the different steps of the cyber kill-chain to the respective layer in a CPS.

In Figure 6.23, we visualize the following mapping of forensic log transitions into the cyber kill-chain. The initial attack-preparing steps `recon` and `weaponization` are unlikely to be detected by our framework at all. As attackers in this phase usually access publicly available information and gather or develop needed tools, they do not interact with our system in a way we could detect. Hence, in other research, these steps are sometimes called `passive scanning`.

If the attack was first seen in a log classified as technical, we likely face a multi-stage attack. Regarding cyber kill-chains for CPSs, this is probably part of the `cyber execution` step as the actual target of an attacker is more likely to be found on the application layer. Therefore, the primary goal of the attacker is not yet reached. Hence, we should be aware that further manipulation may occur in other technical subsystems, e.g., to escalate privileges, or in functional components allowing for deploying the actual payloads.

This transition, from technical log files to functional layer log files, thus, indicates a late stage of an attack as in the cyber kill-chain we are somewhere in the last steps from `cyber execution` to `physical objective`.

In the case, we first see an entry in a functional log and later entries in technical log files, we are probably in earlier attack stages. If an attacker tinkers with technical subsystems after exploiting an application layer component, we assume he just gained access to the system and tries to expand his possibilities. This transition, then, resembles the transitioning from `deliver` to `cyber execution`.

6.3.2 Implementation of Log Aggregation

We use a collection of different aggregators on multiple systems to obtain the required data. While Table 6.8 lists some examples for these data sources along with their log

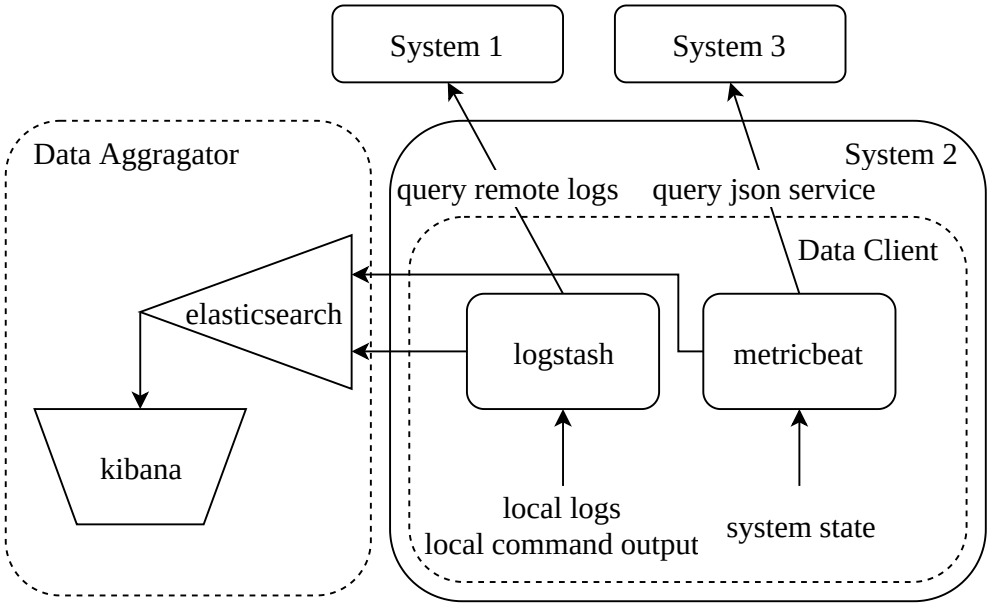

Figure 6.24: Our implementation strategy for aggregating log information.

classification, a more detailed explanation of data gathering for automated risk and anomaly detection and management can be found in Hansch et al. [12]. For this, we use the following tools that are part of the ELK stack [129]: logstash, metricbeat, elasticsearch, and kibana. We continuously list executed software by retrieving a list of current processes on the system using logstash's `exec` input module. In the same way, a list of open ports and processes listening on them corresponds to running local services. Also, application data could be fetched by querying respective applications. Locally accessible logs are monitored using the `file` input module. As logstash has no direct remote connectivity capabilities, we use `smb`, `ssh`, and `nfs` mounts to make respective log files locally accessible. The data client metricbeat delivers general information on the system state. Additionally, we configure its `http` module to periodically query REST endpoints of low resource devices. These respond with their status information in compatible JSON strings.

Figure 6.24, then, shows how we link the used tools of the ELK stack [129] in our architecture. A data aggregation node hosts the elasticsearch database as well as the kibana frontend for data queries and visualization dashboards. Other systems are differentiated according to their resources. With enough computation and storage resources, we directly run the logstash and metricbeat services to deliver local logs, the output of local commands, and the current system state. We call these systems data clients, as they act as worker nodes collecting their own and other systems' information.

Systems with lower resources or access restrictions are accessed by one of these data clients using remote file access or query JSON endpoints returning status information. Hence, for a working data aggregation setup we need at least one data client.

After this data acquisition step, we once need to classify the acquired logs. With this information, analysts may execute the steps outlined in Sections 6.3.1.3 and 6.3.1.4 to identify ongoing attacks and their current progress.

6.3.3 Conclusion

We present a framework for homogenization and aggregation of log files throughout large and complex CPSs. With the proposed method, we support tracking attacks throughout a network of systems without having complete coverage using IDSs. Further, this enables the identification of possible attack paths an attacker used and deriving the attacker's current progress. If early detections are found, we, thus, enable operators to arm defensive mechanisms before an attacker reaches his goal. Finally, this approach enhances forensic analyses by better understanding an attacker's path throughout a CPS network.

6.4 Summary

Contributions

While the previous Chapters 4–5 introduced concrete methods, this chapter focuses on a more abstract view of the larger problem of distributed anomaly detection setups. The provided model of distributed IDSs allows for several more general analyses. In particular, we outlined how specific setups can be analyzed for their performance and which pitfalls may arise during the development and analysis of new detection methods. Apart from abstract considerations, however, it also allows for a practical investigation of current IDS setups and enables optimization of current deployments under scenario-specific constraints. The final concept for the integration of heterogeneous systems into a holistic framework provides first steps towards automated attack entry point detection and consequently a reduced time to fix security vulnerabilities.

Referring to **C4**, we add depth to the often nebulous nature of IDS research. By combining our abstract approach with simulations of real-world deployments and optimization strategies for them, we can derive a much better understanding of the underlying distributed IDS setups and their interplay.

Linkage of previous chapters

Recalling the actual aim of this thesis, we wanted to understand the needs of anomaly detection for CPSs, how to acquire needed data, identify suitable detection methods, and how to link all these parts together in larger systems. We started with our analysis

of the anomaly detection process and derived corresponding needs for data acquisition and methods in Chapter 3. From there, we expanded on two different yet highly practical data acquisition methods in Chapter 4. Further, our previous analysis revealed that we need to put a strong focus on automated methods for the detection of yet unknown attack vectors. The tools and methods presented in Chapter 5 yield two such methods for the actual detection of anomalies. Finally, this chapter built upon the idea of having working anomaly detection systems throughout a more complex CPS installation to highlight their benefits and gaining an understanding of their interplay.

7 Conclusion

In this thesis, we went on a journey throughout the process of adopting, adapting, and analyzing current business IT anomaly and intrusion detection systems in the area of CPSs.

Initially, we postulated four major challenges that have been tackled in the previous chapters (cf. Chapter 1.1).

For the missing data for research and development Chapter 4 presents one method for real-world data acquisition in CPS and another one based on a co-simulation framework in earlier design phases of the system. By this, we establish a framework and workflow to generate meaningful training and test data for anomaly detection systems in industrial settings. Using process-model based simulations, data can be generated on a large scale. We evaluate the data in regard to its usability for state-of-the-art anomaly detection systems. With adequate simulation configurations, it is even possible to simulate a sensor manipulation attack on the model and to derive labeled data.
By this simulation of attacked components, we demonstrate the effectiveness of systems trained on artificial data to detect previously unseen attacks.

A framework for holistic anomaly detection in CPS incorporating data acquisition, method application, and final result handling is presented in Chapter 3. Further, we integrate this framework with security and risk assessments. This makes the whole anomaly detection system compatible with existing standards like IEC62443 [75].

With a specialized detection method shown in Chapter 5, we enhance detection capabilities in CPS under consideration of their specific constraints as resource requirements and used components. Existing frameworks focus on a single Fieldbus protocol or require more detailed knowledge of the CPS itself. Thus, we introduce a uniform method and framework for applying anomaly detection to a variety of Fieldbus protocols. We use stacked denoising autoencoders to derive a feature learning and packet classification method in one step. As the approach is based on the raw byte stream of the network traffic, neither specific protocols nor detailed knowledge of the application is needed. Additionally, we pay attention to creating an efficient framework that can also handle the increased amount of communication in CPSs. Our evaluation on a Secure Water

Treatment dataset using EtherNet/IP and a Modbus dataset shows that we can acquire network packets up to 100 times faster than packet parsing based methods. However, we still achieve precision and recall metrics for longer-lasting attacks of over 99%.

For a better understanding of ongoing attack strategies we investigate the interplay of distributed detection systems in Chapter 6. This provides insights into the overall detection method's system dynamics and allows for recommendations for future installations. As optimization of detection rates is often linked to an increase of false positive rates, we analyze their impact regarding attack detection throughout networks. This enables orchestrated distributed anomaly detection and better forensic analyses of attack strategies. For this purpose, we propose a concept for information aggregation enabling a compound analysis of the involved systems. Using simulations of different configurations, we estimate the impact of detection rates, false positive rates, as well as network topologies on the global system performance. By this study, we provide a method for analyzing the detection capabilities of specific distributed detection system setups allowing for the derivation of appropriate requirements before actual deployment.

7.1 Limitations

For sure, the approaches presented in this thesis come with their own limitations. While we give detailed descriptions of those at the end of each chapter, we shortly summarize the main points.

Incomplete models

The process described for deriving anomaly and intrusion detection rules definitely depends on the initial system model. Aspects that have been missed throughout the modeling phase cannot be caught later on. Also, while we derive rules to surveil the system behavior, the checks may still be circumvented by specialized attacks.

Unrealistic simulation

While we describe a method for direct data acquisition in already running real-world systems, we are aware that CPS operators may be reluctant to implement these into their systems. Therefore, an alternative method based on the simulation of the physical process is presented. The results of this simulation can only be as good as the simulation itself. For this, we assume that the design and development phase of the CPS already required the implementation of a suitable process simulation. As this simulation must be close enough to the final deployment that possible problems rise early, we assume that the final CPS has only little differences to the simulation.

With our proposed anomaly and intrusion detection method, we focus specifically on closed-source protocols that cannot be parsed efficiently in real-time applications. If,

on the other hand, such a deep packet inspection was possible, other methods may be more suitable for detection. While we evaluated our approach with the largest dataset currently existing to our knowledge, there is still the potential we were just lucky. As described in Chapter 2, machine-learning based methods always only derive an approximation of the desired decision function. Hence, the application of this approximation outside of its scope, i.e., extrapolation, may yield unexpected results.

Uncertainties of ML

Finally, our analysis of distributed detection systems comes with many assumptions already described throughout the chapter. However, these were mainly taken to tighten the scope of the analysis and allowing for an evaluation of the concept. Most assumptions, like the uniform spreading of attacks throughout a network or the uniformity of the detection systems, can be extended and enhanced in future research. While we provide a first concept for the identification of entry points in distributed setups, still further research is needed to validate and evaluate this idea.

Environment assumptions

7.2 Future Research

Connecting to the previous paragraph, the analysis of overall anomaly and intrusion detection systems carries much potential for extensions. By adding non-uniform spreading of attacks, we can include the difficulty of an attacker using a specific connection in the system. Such information can, for example, be derived from a security and risk analysis similar to the concept described in Chapter 3. Further, we can allow a mixture of different detection systems with different properties. These could even be considered changing over time. By adding an exponential decay in the detection rate over time, an ongoing outdating of current attack signatures without updates may be simulated. In the end, this can lead to an even more realistic detection simulation. Combined with the already investigated optimization strategies, questions for optimal positioning of sensors, optimal update frequencies, or even a dynamic movement strategy for detection sensors throughout the system can be tackled.

Refined simulation model

In a broader sense, this leads to a better understanding of the needs of anomaly and intrusion detection for CPSs. In fact, every intrusion detection system is developed around three parts: the data analyzed, the method used, and the architecture or framework used for processing. These three items are as described in Chapter 1.3 only building blocks for efficient anomaly detection in industrial use cases.

To finally overcome cumbersome manual tasks and to ease the detection of yet unknown attack schemes, we see two integral parts to be tackled in the near future: the

automated development of *normal* behavior models as well as efforts for more human-centric system designs.

Assistive information

For example, recent advances in natural language processing may enable us to derive lists of assets from concept and design documents automatically. Combining this information with common system architecture schemes and a semantic understanding of the design documents, the often tedious but central task of system modeling and understanding can be assisted. This system model can then be combined with publicly available information from news bulletins and Common Vulnerabilities and Exposuress (CVEs) to generate a fast-forward pipeline introducing automated risk assessments and corresponding security measure deduction even for small- and medium-sized economies.

Human-centric design

Another often neglected realm is the missing integration or consideration of humans-in-the-loop in anomaly detection systems. Between security analysts, much information is currently shared on social media, e.g., twitter, on news feeds, and blogs as well by CVEs and security alerts. However, only a few anomaly detection systems supply or even consider this information during analysis. Overthinking the representation of the generated alert messages may enable security analysts for a quicker understanding of root causes and attack courses. This embedding of anomaly detection systems in its operator's world leads us towards a more human-centric detection system.

Outcome

Throughout this thesis, we provided an in-depth look at the building blocks for anomaly detection for cyber-physical systems. With contributions as the identification of suitable IDSs, the intelligent acquisition of data in CPS contexts, a protocol-agnostic detection method, and the understanding of complex IDS network characteristics, we provide tools for an easier integration of IDSs into CPSs. In all these methods, we focus on making existing solutions better suited for the specific environment and constraints of CPSs. Doing so, we enable CPS operators to deploy IDSs quicker and more easily as part of their security concepts.

In a constantly changing IT threat landscape, relying solely on one countermeasure seems to be a safe route to trouble. IDSs can, therefore, always only be part of a larger protection concept. Nonetheless, this thesis allows CPS operators and researchers to integrate IDSs into this concept. Thereby, we add another layer of security to highly vulnerable systems.

Bibliography

[1] Edward A. Lee. Cyber Physical Systems: Design Challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369. IEEE, may 2008. ISBN 978-0-7695-3132-8. doi: 10.1109/ISORC.2008.25. URL `http://ieeexplore.ieee.org/document/4519604/`.

[2] S D Antón, D Fraunholz, C Lipps, F Pohl, M Zimmermann, and H D Schotten. Two decades of SCADA exploitation: A brief history. In *2017 IEEE Conference on Application, Information and Network Security (AINS)*, pages 98–104, nov 2017. ISBN 9781538607251. doi: 10.1109/AINS.2017.8270432.

[3] Ralph Langner. To Kill a Centrifuge: A Technical Analysis of What Stuxnet's Creators Tried to Achieve. Technical report, The Langner Group, 2013.

[4] Robert M. Lee, Michael J. Assante, and Tim Conway. Analysis of the Cyber Attack on the Ukrainian Power Grid. Technical report, Electricity Information Sharing and Analysis Center (E-ISAC), 2016.

[5] Martin Brunner, Hans Hofinger, Christoph Krauß, Christopher Roblee, Peter Schoo, and Sascha Todt. Infiltrating Critical Infrastructures with Next-Generation Attacks: W32. Stuxnet as a Showcase Threat. Technical report, Fraunhofer Research Institution AISEC, Munich, 2010.

[6] G. P. H. Sandaruwan, P. S. Ranaweera, and V. A. Oleshchuk. PLC security and critical infrastructure protection. In *2013 IEEE 8th International Conference on Industrial and Information Systems*, pages 81–85, 2013. ISBN 9781479909100. doi: 10.1109/ICIInfS.2013.6731959.

[7] Earlence Fernandes, Amir Rahmati, Kevin Eykholt, and Atul Prakash. Internet of Things Security Research: A Rehash of Old Ideas or New Intellectual Challenges? *IEEE Security and Privacy*, 15(4):79–84, 2017. ISSN 15584046. doi: 10.1109/MSP.2017.3151346.

[8] Gerhard Hansch, Peter Schneider, and Sven Plaga. Packet-wise compression and forwarding of industrial network captures. In *Proceedings of the 2017 IEEE 9th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2017*, volume 1, pages 66–70, 2017. ISBN 9781538606971. doi: 10.1109/IDAACS.2017.8095051.

[9] Peter Schneider and Alexander Giehl. Realistic Data Generation for Anomaly Detection in Industrial Settings Using Simulations. In *Computer Security*, pages 119–134, Cham, 2019. Springer International Publishing. ISBN 978-3-030-12786-2.

[10] Peter Schneider and Konstantin Böttinger. High-Performance Unsupervised Anomaly Detection for Cyber-Physical System Networks. *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy - CPS-SPC '18*, pages 1–12, 2018. ISSN 15437221. doi: 10.1145/3264888.3264890.

[11] Gerhard Hansch, Peter Schneider, Kai Fischer, and Konstantin Böttinger. A Unified Architecture for Industrial IoT Security Requirements in Open Platform Communications. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2019-Septe:325–332, 2019. ISSN 19460759. doi: 10.1109/ETFA.2019.8869524.

[12] Gerhard Hansch, Peter Schneider, and Gerd Stefan Brost. Deriving Impact-driven Security Requirements and Monitoring Measures for Industrial IoT. In *Proceedings of the 5th on Cyber-Physical System Security Workshop - CPSS '19*, pages 37–45, New York, New York, USA, 2019. ACM Press. ISBN 9781450367875. doi: 10.1145/3327961.3329528. URL `http://dl.acm.org/citation.cfm?doid=3327961.3329528`.

[13] Alexander Giehl, Peter Schneider, Maximilian Busch, Florian Schnoes, Robin Kleinwort, and Michael F Zaeh. Edge-computing enhanced privacy protection for industrial ecosystems in the context of SMEs. In *2019 12th CMI Conference on Cybersecurity and Privacy (CMI)*, pages 1–6, 2019. ISBN 9781728128566. doi: 10.1109/CMI48017.2019.8962138.

[14] Peter Schneider. Do's and Don'ts of Distributed Intrusion Detection for Industrial Network Topologies. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3222–3231, 2020. ISBN 9781728108582. doi: 10.1109/BigData47090.2019.9006117.

[15] Shuang Huang, Chun Jie Zhou, Shuang Hua Yang, and Yuan Qing Qin. Cyber-physical system security for networked industrial processes. *International Journal of Automation and Computing*, 12(6):567–578, 2015. ISSN 17518520. doi: 10.1007/s11633-015-0923-9.

[16] Gaurav Tandon. *Machine Learning for Host-based Anomaly Detection*. Dissertation, Florida Institute of Technology, 2008.

[17] Jan Vavra and Martin Hromada. Evaluation of anomaly detection based on classification in relation to SCADA. In *ICMT 2017 - 6th International Conference on Military Technologies*, pages 330–334, 2017. ISBN 9781538619889. doi: 10.1109/MILTECHS.2017.7988779.

[18] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly Detection: A Survey. *ACM Comput. Surv.*, 41(3):1–58, 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541882.

[19] Wayne Wolf. Cyber-physical Systems. *Computer*, 42(3):88–89, mar 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.81. URL `http://ieeexplore.ieee.org/document/4803901/`.

[20] A. A. Letichevsky, O. O. Letychevskyi, V. G. Skobelev, and V. A. Volkov. Cyber-Physical Systems. *Cybernetics and Systems Analysis*, 53(6):821–834, nov 2017. ISSN 1060-0396. doi: 10.1007/s10559-017-9984-9. URL `http://link.springer.com/10.1007/s10559-017-9984-9`.

[21] Christopher Greer, Martin Burns, David Wollman, and Edward Griffor. Cyber-physical systems and internet of things. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, mar 2019. URL `https://ieeexplore.ieee.org/document/8591610/https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1900-202.pdf`.

[22] Martin Roesch. Snort — lightweight intrusion detection for networks. In *Proceedings of LISA '99: 13th Systems Administration Conference*, pages 229–238, 1999.

[23] Robert Udd, Mikael Asplund, Simin Nadjm-Tehrani, Mehrdad Kazemtabrizi, and Mathias Ekstedt. Exploiting Bro for Intrusion Detection in a SCADA System. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, pages 44–51, New York, NY, USA, may 2016. ACM. ISBN

9781450342889. doi: 10.1145/2899015.2899028. URL https://dl.acm.org/doi/10.1145/2899015.2899028.

[24] Kevin Wong, Craig Dillabaugh, Nabil Seddigh, and Biswajit Nandy. Enhancing Suricata intrusion detection system for cyber security in SCADA networks. In *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–5. IEEE, apr 2017. ISBN 978-1-5090-5538-8. doi: 10.1109/CCECE.2017.7946818. URL http://ieeexplore.ieee.org/document/7946818/.

[25] Ali Anafcheh. *Intrusion Detection with OSSEC*. Bachelor's thesis, South-Eastern Finland University of Applied Sciences, 2018.

[26] Wazuh Inc. wazuh, 2015. URL https://wazuh.com/.

[27] Lionel N. Tidjon, Marc Frappier, and Amel Mammar. Intrusion Detection Systems: A Cross-Domain Overview. *IEEE Communications Surveys & Tutorials*, PP(3):1–1, 2019. doi: 10.1109/comst.2019.2922584.

[28] Wei-Yin Loh. Classification and regression trees. *WIREs Data Mining and Knowledge Discovery*, 1(1):14–23, jan 2011. ISSN 1942-4787. doi: 10.1002/widm.8. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.8.

[29] Michael W. Berry, Azlinah Mohamed, and Bee Wah Yap. *Supervised and Unsupervised Learning for Data Science*. Number January in Unsupervised and Semi-Supervised Learning. Springer International Publishing, Cham, 2020. ISBN 978-3-030-22474-5. doi: 10.1007/978-3-030-22475-2. URL http://link.springer.com/10.1007/978-3-030-22475-2.

[30] B. Bencsáth, G. Pék, L. Buttyán, and M. Félegyházi. Duqu: A Stuxnet-like malware found in the wild. *CrySyS Lab Technical Report*, 14:1–60, 2011.

[31] B Bencsáth, G Pék, L Buttyán, and M Felegyhazi. sKyWIper (a.k.a. Flame a.k.a. Flamer): A complex malware for targeted attacks. Technical report, CrySyS Lab, 2012. URL www.crysys.hu.

[32] Hamilton Turner, Jules White, Jaime A Camelio, Christopher Williams, Brandon Amos, and Robert Parker. Bad Parts: Are Our Manufacturing Systems at Risk of Silent Cyberattacks? *IEEE Security and Privacy*, 13(3):40–47, 2015. ISSN 15407993. doi: 10.1109/MSP.2015.60.

[33] Danny Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, mar 1983. ISSN 0018-9448. doi: 10.1109/TIT.1983.1056650. URL `http://ieeexplore.ieee.org/document/1056650/`.

[34] Jean-Loup Gailly and Mark Adler. Zlib compression library, 2004. URL `http://www.dspace.cam.ac.uk/handle/1810/3486`.

[35] Jacob Ziv and Abraham Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 1977. ISSN 15579654. doi: 10.1109/TIT.1977.1055714.

[36] Antoine Lemay and José M Fernandez. Providing SCADA network data sets for intrusion detection research. In *9th Workshop on Cyber Security Experimentation and Test (CSET 16)*, Austin, TX, 2016. USENIX Association. URL `https://www.usenix.org/conference/cset16/workshop-program/presentation/lemay`.

[37] Jiexin Zhang, Shaoduo Gan, Xiaoxue Liu, and Peidong Zhu. Intrusion detection in SCADA systems by traffic periodicity and telemetry analysis. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 318–325. IEEE, jun 2016. ISBN 978-1-5090-0679-3. doi: 10.1109/ISCC.2016.7543760.

[38] Rohan Chabukswar, Bruno Sinopoli, Gabor Karsai, Annarita Giani, Himansh Neema, and Andrew Davis. Simulation of Network Attacks on SCADA Systems. In *First Workshop on Secure Control Systems*, pages 587–592, 2010. URL `https://ptolemy.berkeley.edu/projects/truststc/conferences/10/CPSWeek/presentations/RohanChabukswar.pdf`.

[39] Daniele Antonioli and Nils Ole Tippenhauer. MiniCPS. In *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy - CPS-SPC '15*, pages 91–100, New York, New York, USA, 2015. ACM Press. ISBN 9781450338271. doi: 10.1145/2808705.2808715. URL `http://dl.acm.org/citation.cfm?doid=2808705.2808715`.

[40] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA Offline Intrusion Detection Evaluation. *Computer Networks*, 34(4):579–595, 2000. ISSN 13891286. doi: 10.1016/S1389-1286(00)00139-0.

[41] Antoine Lemay, José M. Fernandez, and Scott Knight. A Modbus command and control channel. In *2016 Annual IEEE Systems Conference (SysCon)*, pages 1–6. IEEE, apr 2016. ISBN 978-1-4673-9519-9. doi: 10.1109/SYSCON.2016.7490631. URL `http://ieeexplore.ieee.org/document/7490631/`.

[42] NETRESEC. Capture files from 4SICS Geek Lounge, 2017. URL `https://www.netresec.com/?page=PCAP4SICS`.

[43] Hannes Holm, Martin Karresand, Arne Vidström, and Erik Westring. A Survey of Industrial Control System Testbeds. In *Nordic Conference on Secure IT Systems*, pages 11–26. Springer, 2015. ISBN 9783319265025. doi: 10.1007/978-3-319-26502-5.

[44] Richard Candell Jr., Timothy A. Zimmerman, and Keith A. Stouffer. An Industrial Control System Cybersecurity Performance Testbed. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, dec 2015. URL `https://nvlpubs.nist.gov/nistpubs/ir/2015/NIST.IR.8089.pdf`.

[45] Stephen McLaughlin, Charalambos Konstantinou, Xueyang Wang, Lucas Davi, Ahmad Reza Sadeghi, Michail Maniatakos, and Ramesh Karri. The Cybersecurity Landscape in Industrial Control Systems. *Proceedings of the IEEE*, 104(5):1039–1057, 2016. ISSN 00189219. doi: 10.1109/JPROC.2015.2512235.

[46] 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, 2015. ISBN 9781450334488. doi: 10.1145/2732198.2732200.

[47] Stanislav Ponomarev and Travis Atkison. Industrial Control System Network Intrusion Detection by Telemetry Analysis. *IEEE Transactions on Dependable and Secure Computing*, 13(2):252–260, 2016. ISSN 15455971. doi: 10.1109/TDSC.2015.2443793.

[48] Naman Govil, Anand Agrawal, and Nils Ole Tippenhauer. On ladder logic bombs in industrial control systems. In *Computer Security, Springer*, volume 10683 LNCS, pages 110–126, 2017. ISBN 9783319728162. doi: 10.1007/978-3-319-72817-9_8.

[49] Ali Abbasi and Majid Hashemi. Ghost in the PLC Designing an Undetectable Programmable Logic Controller Rootkit via Pin Control Attack. *Blackhat Europe*,

pages 1–35, 2016. URL https://www.blackhat.com/docs/eu-16/materials/eu-16-Abbasi-Ghost-In-The-PLC-Designing-An-Undetectable-Programmable-Logic-Controller-pdf.

[50] Amit Kleinmann, Ori Amichay, Avishai Wool, David Tenenbaum, Ofer Bar, and Leonid Lev. Stealthy deception attacks against SCADA systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10683 LNCS, pages 93–109, 2017. ISBN 9783319728162. doi: 10.1007/978-3-319-72817-9_7.

[51] Abdulmohsen Almalawi, Adil Fahad, Zahir Tari, Abdullah Alamri, Rayed Alghamdi, and Albert Y. Zomaya. An Efficient Data-Driven Clustering Technique to Detect Attacks in SCADA Systems. *IEEE Transactions on Information Forensics and Security*, 11(5):893–906, 2016. ISSN 15566013. doi: 10.1109/TIFS.2015.2512522.

[52] Fabio Pasqualetti, Florian Dorfler, and Francesco Bullo. Attack detection and identification in cyber-physical systems. *IEEE Transactions on Automatic Control*, 58(11):2715–2729, 2013. ISSN 00189286. doi: 10.1109/TAC.2013.2266831.

[53] Piroska Haller and Bela Genge. Using Sensitivity Analysis and Cross-Association for the Design of Intrusion Detection Systems in Industrial Cyber-Physical Systems. *IEEE Access*, 5:9336–9347, 2017. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2703906.

[54] Sasanka Potluri, Christian Diedrich, and Girish Kumar Reddy Sangala. Identifying false data injection attacks in industrial control systems using artificial neural networks. In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pages 1–8, 2017. ISBN 9781509065059. doi: 10.1109/ETFA.2017.8247663.

[55] Matthias Eckhart and Andreas Ekelhart. A Specification-Based State Replication Approach for Digital Twins. In *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*, volume 12 of *CPS-SPC '18*, pages 36–47, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450359924. doi: 10.1145/3264888.3264892. URL https://doi.org/10.1145/3264888.3264892.

[56] Alexander Giehl and Norbert Wiedermann. Security Verification of Third Party Design Files in Manufacturing. In *Proceedings of the 2018 10th International*

Conference on Computer and Automation Engineering, pages 166–173, Brisbane, 2018. Association for Computing Machinery. ISBN 9781450364935. doi: 10.1145/3192975.3192984.

[57] Justin M Beaver, Raymond C. Borges-Hink, and Mark A Buckner. An evaluation of machine learning methods to detect malicious SCADA communications. In *2013 12th International Conference on Machine Learning and Applications*, volume 2, pages 54–59, 2013. ISBN 978-0-7695-5144-9. doi: 10.1109/ICMLA.2013.105.

[58] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. *arXiv preprint arXiv:1802.09089*, pages 18–21, feb 2018. URL `http://arxiv.org/abs/1802.09089`.

[59] Y. Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. ISSN 00189219. doi: 10.1109/5.726791. URL `http://ieeexplore.ieee.org/document/726791/`.

[60] Mark Nicolett and Kelly M Kavanagh. Magic quadrant for security information and event management evaluation criteria definitions. Technical Report May, Gartner Research, 2011.

[61] Idan Amit, John Matherly, William Hewlett, Zhi Xu, Yinnon Meshi, and Yigal Weinberger. Machine Learning in Cyber-Security - Problems, Challenges and Data Sets. *arXiv preprint arXiv:1812.07858*, dec 2018. URL `http://arxiv.org/abs/1812.07858`.

[62] Alireza Sadighian, Saman Taghavi Zargar, Jose M. Fernandez, and Antoine Lemay. Semantic-based context-aware alert fusion for distributed Intrusion Detection Systems. In *2013 International Conference on Risks and Security of Internet and Systems (CRiSIS)*, pages 1–6. IEEE, oct 2013. ISBN 978-1-4799-3488-1. doi: 10.1109/CRiSIS.2013.6766352. URL `http://ieeexplore.ieee.org/document/6766352/`.

[63] Hichem Sedjelmaci, Sidi Mohammed Senouci, and Mohamad Al-Bahri. A lightweight anomaly detection technique for low-resource IoT devices: A game-theoretic methodology. In *2016 IEEE International Conference on Communica-*

tions (ICC), pages 1–6. IEEE, may 2016. ISBN 978-1-4799-6664-6. doi: 10.1109/ICC.2016.7510811. URL `http://ieeexplore.ieee.org/document/7510811/`.

[64] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and other botnets. *Computer*, 50(7):80–84, 2017. ISSN 00189162. doi: 10.1109/MC.2017.201.

[65] Markus Graube, Stephan Hensel, Chris Iatrou, and Leon Urbas. Information models in OPC UA and their advantages and disadvantages. In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pages 1–8, 2018. ISBN 9781509065059. doi: 10.1109/ETFA.2017.8247691.

[66] N Jazdi. Cyber physical systems in the context of Industry 4.0. In *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, pages 1–4. IEEE, may 2014. ISBN 978-1-4799-3732-5. doi: 10.1109/AQTR.2014.6857843. URL `http://ieeexplore.ieee.org/document/6857843/`.

[67] Jahanzaib Imtiaz and Jurgen Jasperneite. Scalability of OPC-UA down to the chip level enables "internet of Things". In *IEEE International Conference on Industrial Informatics (INDIN)*, pages 500–505, 2013. ISBN 9781479907526. doi: 10.1109/INDIN.2013.6622935.

[68] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. A Dataset to Support Research in the Design of Secure Water Treatment Systems. In Grigore Havarneanu, Roberto Setola, Hypatia Nassopoulos, and Stephen Wolthusen, editors, *Critical Information Infrastructures Security*, pages 88–99, Cham, 2017. Springer International Publishing. ISBN 978-3-319-71368-7. doi: 10.1007/978-3-319-71368-7_8. URL `http://link.springer.com/10.1007/978-3-319-71368-7{_}8`.

[69] Ahmad M Karimi, Quamar Niyaz, Weiqing Sun, Ahmad Y Javaid, and Vijay K Devabhaktuni. Distributed network traffic feature extraction for a real-time IDS. In *2016 IEEE International Conference on Electro Information Technology (EIT)*, pages 0522–0526. IEEE, may 2016. ISBN 978-1-4673-9985-2. doi: 10.1109/EIT.2016.7535295. URL `http://ieeexplore.ieee.org/document/7535295/`.

[70] Obinna Igbe, Ihab Darwish, and Tarek Saadawi. Distributed Network Intrusion Detection Systems: An Artificial Immune System Approach. In *2016 IEEE First International Conference on Connected Health: Applications, Systems and*

Engineering Technologies (CHASE), pages 101–106. IEEE, jun 2016. ISBN 978-1-5090-0943-5. doi: 10.1109/CHASE.2016.36. URL `http://ieeexplore.ieee.org/document/7545821/`.

[71] Ahmad W. Al-Dabbagh, Yuzhe Li, and Tongwen Chen. An Intrusion Detection System for Cyber Attacks in Wireless Networked Control Systems. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(8):1049–1053, aug 2018. ISSN 1549-7747. doi: 10.1109/TCSII.2017.2690843. URL `https://ieeexplore.ieee.org/document/7891998/`.

[72] Hao Chen, John A. Clark, Siraj A. Shaikh, Howard Chivers, and Philip Nobles. Optimising IDS sensor placement. In *ARES 2010 - 5th International Conference on Availability, Reliability, and Security*, pages 315–320, 2010. ISBN 9780769539652. doi: 10.1109/ARES.2010.92.

[73] Steven Noel and Sushil Jajodia. Optimal IDS Sensor Placement and Alert Prioritization Using Attack Graphs. *Journal of Network and Systems Management*, 16(3):259–275, sep 2008. ISSN 1064-7570. doi: 10.1007/s10922-008-9109-x. URL `http://link.springer.com/10.1007/s10922-008-9109-x`.

[74] Shannon L. Isovitsch and Jeanne M. VanBriesen. Sensor Placement and Optimization Criteria Dependencies in a Water Distribution System. *Journal of Water Resources Planning and Management*, 134 (2):186–196, mar 2008. ISSN 0733-9496. doi: 10.1061/(ASCE)0733-9496(2008)134:2(186). URL `http://ascelibrary.org/doi/10.1061/{%}28ASCE{%}290733-9496{%}282008{%}29134{%}3A2{%}28186{%}29`.

[75] IEC. IEC 62443-2-1 Industrial communication networks – Network and system security - Part 2-1: Establishing an industrial automation and control system security program, 2010.

[76] Jack A. Jones. An Introduction to Factor Analysis of Information Risk (FAIR). Technical report, Risk Management Insight, 2005. URL `http://www.riskmanagementinsight.com`.

[77] Richard Caralli, James F. Stevens, Lisa R. Young, and William R. Wilson. Introducing OCTAVE Allegro : Improving the Information Security Risk Assessment Process. Technical report, Software Engineering Institute, 2007.

[78] M. A. Amutio, J. Candau, and J. Antonio Mañas. MAGERIT - version 3.0. Methodology for Information Systems Risk Analysis and Management - The

Method. Technical report, Ministry of Finance and Public Administration, Spain, 2014.

[79] Daniel Angermeier, Alexander Nieding, Jörn Eichler, and Fraunhofer Aisec. Systematic Identification of Security Goals and Threats in Risk Assessment. *Softwaretechnik-Trends*, 36(3), 2016.

[80] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. *Lecture Notes in Computer Science*, 3494:19–35, 2005. ISSN 03029743. doi: 10.1007/11426639_2.

[81] Albert Treytl, Thilo Sauter, and Christian Schwaiger. Security Measures for Industrial Fieldbus Systems - State of the Art and Solutions for IP-based Approaches. In *IEEE International Workshop on Factory Communication Systems, 2004. Proceedings.*, pages 201–209, 2004. doi: 10.1109/WFCS.2004.1377709.

[82] Markus Runde, Christopher Tebbe, and Karl Heinz Niemann. Performance evaluation of an IT security layer in real-time communication. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pages 1–4, 2013. ISSN 19460740. doi: 10.1109/ETFA.2013.6648104.

[83] Felix Wieczorek, Christoph Krauß, Frank Schiller, and Claudia Eckert. Towards secure fieldbus communication. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7612 LNCS:149–160, 2012. ISSN 03029743. doi: 10.1007/978-3-642-33678-2_13.

[84] Johan Åkerberg and Mats Björkman. Introducing security modules in PROFINET IO. *ETFA 2009 - 2009 IEEE Conference on Emerging Technologies and Factory Automation*, pages 1–8, 2009. doi: 10.1109/ETFA.2009.5347205.

[85] H Boterenbrood. CANopen high-level protocol for CAN-bus. Technical report, Nikhef, Amsterdam, 2000.

[86] Chih-Ta Lin, Sung-Lin Wu, and Mei-Lin Lee. Cyber attack and defense on industry control systems. In *2017 IEEE Conference on Dependable and Secure Computing*, pages 524–526, 2017. doi: 10.1109/DESEC.2017.8073874.

[87] Robert Love. Kernel Korner: Intro to Inotify. *Linux Journal*, 2005(139):8, 2005.

[88] Suricata. The Next Generation Intrusion Detection System, 2018. URL `https://suricata-ids.org/`.

[89] Dina Hadziosmanovic, Robin Sommer, Emmanuele Zambon, and Pieter Hartel. Through the Eye of the PLC: Semantic Security Monitoring for Industrial Processes. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 126–135, New Orleans, Louisiana, USA, 2014. Association for Computing Machinery. ISBN 9781450330053. doi: 10.1145/2664243.2664277.

[90] Philippe Biondi. Network packet manipulation with Scapy, 2005. URL `https://scapy.net/conf/scapy{_}hack.lu.pdf`.

[91] Google Inc. Protocol Buffers - Google's data interchange format, 2008. URL `https://github.com/google/protobuf`.

[92] Alexander Giehl. *Development of a Co-Simulation framework to analyse attacks and their impact on Smart Grid*. Master's thesis, Technische Universität München, 2013.

[93] Peter Fritzson, Peter Aronsson, Adrian Pop, Håkan Lundvall, Kaj Nyström, Levon Saldamli, David Broman, and Anders Sandholm. OpenModelica – A Free Open-Source Environment for System Modeling, Simulation, and Teaching. *Proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design*, pages 1588–1595, 2006.

[94] Gerald Reichl. WasteWater a library for modelling and simulation of wastewater treatment plants in modelica. In *The 3rd International Modelica Conference*, pages 171–176, 2003. doi: 10.1.1.76.5709. URL `https://www.modelica.org/events/Conference2003/papers/h05{_}Reichl.pdf`.

[95] Francesco Casella and Alberto Leva. Modelica open library for power plant simulation: design and experimental validation. *Proceedings of the 3rd International Modelica Conference*, pages 41–50, 2003.

[96] Gustavo J. A. M. Carneiro. NS-3: Network Simulator 3. In *UTM Lab Meeting April*, 2010. ISBN 9783540773405. URL `https://www2.nsnam.org/tutorials/NS-3-LABMEETING-1.pdf`.

[97] Modbus-IDA. MODBUS Messaging on TCP/IP Implementation Guide 1.0a, 2004.

[98] Marco Bonvini and Alberto Leva. A Modelica Library for Industrial Control Systems. In *Proceedings of the 9th International MODELICA Conference,*

September 3-5, 2012, Munich, Germany, volume 76, pages 477–484, 2012. doi: 10.3384/ecp12076477.

[99] Robin Sommer and Vern Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316, Berkeley/Oakland, CA, 2010. IEEE. ISBN 978-1-4244-6894-2. doi: 10.1109/SP.2010.25. URL `http://ieeexplore.ieee.org/document/5504793/`.

[100] Charles Hornig. A Standard for the Transmission of IP Datagrams over Ethernet Networks. Technical report, Network Working Group, apr 1984. URL `https://www.rfc-editor.org/info/rfc0894`.

[101] Yoshua Bengio. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009. ISSN 1935-8237. doi: 10.1561/2200000006. URL `http://www.nowpublishers.com/article/Details/MAL-006`.

[102] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, jul 2006. ISSN 0036-8075. doi: 10.1126/science.1127647. URL `https://www.sciencemag.org/lookup/doi/10.1126/science.1127647`.

[103] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 1096–1103, Helsinki, Finland, 2008. ACM Press. ISBN 9781605582054. doi: 10.1145/1390156.1390294. URL `http://portal.acm.org/citation.cfm?doid=1390156.1390294`.

[104] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, jan 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492. URL `https://dl.acm.org/doi/10.1145/1327452.1327492`.

[105] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking State-of-the-Art Deep Learning Software Tools. In *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, pages 99–104. IEEE, nov 2016. ISBN 978-1-5090-3555-7. doi: 10.1109/CCBD.2016.029. URL `http://arxiv.org/abs/1608.07249http://ieeexplore.ieee.org/document/7979887/`.

[106] Gerald Combs. Wireshark, 2006. URL `https://www.wireshark.org`.

[107] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, 2016. USENIX Association. ISBN 978-1-931971-33-1.

[108] Dor Green. pyshark, 2013. URL `http://kiminewt.github.io/pyshark/`.

[109] Arturo Filastò. pypcap, 2004. URL `https://github.com/pynetwork/pypcap`.

[110] Daniele Antonioli, Hamid Reza Ghaeini, Sridhar Adepu, Martin Ochoa, and Nils Ole Tippenhauer. Gamifying ICS Security Training and Research: Design, Implementation, and Results of S3. In *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 93–102, New York, NY, USA, nov 2017. Association for Computing Machinery. ISBN 9781450353946. doi: 10.1145/3140241.3140253. URL `https://dl.acm.org/doi/10.1145/3140241.3140253`.

[111] Elissa M Redmiles. "Should I Worry?" A Cross-Cultural Examination of Account Security Incident Response. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 920–934. IEEE, may 2019. ISBN 978-1-5386-6660-9. doi: 10.1109/SP.2019.00059. URL `https://ieeexplore.ieee.org/document/8835359/`.

[112] G.C. Tjhai, Maria Papadaki, S.M. Furnell, and N.L. Clarke. Investigating the problem of IDS false alarms: An experimental study using Snort. In *Proceedings of The Ifip Tc 11 23rd International Information Security Conference*, volume 278, pages 253–267. Springer US, Boston, MA, 2008. ISBN 9780387096988. doi: 10.1007/978-0-387-09699-5_17. URL `http://link.springer.com/10.1007/978-0-387-09699-5{_}17`.

[113] Yuxin Meng and Lam-for Kwok. Adaptive False Alarm Filter Using Machine Learning in Intrusion Detection. In *Advances in Intelligent and Soft Computing*, volume 124, pages 573–584. Springer, 2011. ISBN 9783642256578. doi: 10.1007/978-3-642-25658-5_68. URL `http://link.springer.com/10.1007/978-3-642-25658-5{_}68`.

[114] Adam Hahn, Roshan K. Thomas, Ivan Lozano, and Alvaro Cardenas. A multi-layered and kill-chain based security analysis framework for cyber-physical systems. *International Journal of Critical Infrastructure Protection*, 11:39–50, dec 2015. ISSN 18745482. doi: 10.1016/j.ijcip.2015.08.003. URL `https://linkinghub.elsevier.com/retrieve/pii/S187454821500061X`.

[115] Karsten Nohl and Jakob Lell. BadUSB - On accessories that turn evil. In *Black Hat USA*, 2014. URL `https://srlabs.de/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf`.

[116] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, dec 1959. ISSN 0029-599X. doi: 10.1007/BF01386390. URL `http://link.springer.com/10.1007/BF01386390`.

[117] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz— Open Source Graph Drawing Tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer, 2002. doi: 10.1007/3-540-45848-4_57. URL `http://link.springer.com/10.1007/3-540-45848-4{_}57`.

[118] Asuka Terai, Shingo Abe, Shoya Kojima, Yuta Takano, and Ichiro Koshijima. Cyber-Attack Detection for Industrial Control System Monitoring with Support Vector Machine Based on Communication Profile. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 132–138. IEEE, apr 2017. ISBN 978-1-5386-2244-5. doi: 10.1109/EuroSPW.2017.62. URL `http://ieeexplore.ieee.org/document/7966982/`.

[119] Imtiaz Ullah and Qusay H Mahmoud. A hybrid model for anomaly-based intrusion detection in SCADA networks. In *2017 IEEE International Conference on Big Data (Big Data)*, volume 2018-Janua, pages 2160–2167. IEEE, dec 2017. ISBN 978-1-5386-2715-0. doi: 10.1109/BigData.2017.8258164. URL `http://ieeexplore.ieee.org/document/8258164/`.

[120] Moshe Kravchik and Asaf Shabtai. Detecting Cyber Attacks in Industrial Control Systems Using Convolutional Neural Networks. In *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 72–83, New York, NY, USA, jan 2018. ACM. ISBN 9781450359924. doi: 10.1145/3264888.3264896. URL `https://dl.acm.org/doi/10.1145/3264888.3264896`.

[121] Ekta Aggarwal, Mehdi Karimibiuki, Karthik Pattabiraman, and André Ivanov. CORGIDS. In *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 24–35, New York, NY, USA, jan 2018. ACM. ISBN 9781450359924. doi: 10.1145/3264888.3264893. URL `https://dl.acm.org/doi/10.1145/3264888.3264893`.

[122] Sasanka Potluri, Navin Francis Henry, and Christian Diedrich. Evaluation of hybrid deep learning techniques for ensuring security in networked control systems. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, sep 2017. ISBN 978-1-5090-6505-9. doi: 10.1109/ETFA.2017.8247662. URL `http://ieeexplore.ieee.org/document/8247662/`.

[123] Weizhong Yan, Lalit K. Mestha, and Masoud Abbaszadeh. Attack Detection for Securing Cyber Physical Systems. *IEEE Internet of Things Journal*, 6(5): 8471–8481, oct 2019. ISSN 2327-4662. doi: 10.1109/JIOT.2019.2919635. URL `https://ieeexplore.ieee.org/document/8728231/`.

[124] Jakob Rieck. Attacks on Fitness Trackers Revisited: A Case-Study of Unfit Firmware Security. *arXiv preprint arXiv:1604.03313*, pages 1–12, apr 2016. URL `http://arxiv.org/abs/1604.03313`.

[125] Arthur Juliani. Simple Reinforcement Learning with Tensorflow, 2016. URL `https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-ar`

[126] Sung Jun Son and Youngmi Kwon. Performance of ELK stack and commercial system in security log analysis. In *2017 IEEE 13th Malaysia International Conference on Communications (MICC)*, volume 2017-Novem, pages 187–190. IEEE, nov 2017. ISBN 978-1-5386-3132-4. doi: 10.1109/MICC.2017.8311756. URL `http://ieeexplore.ieee.org/document/8311756/`.

[127] Explosion AI. spaCy: Industrial-strength Natural Language Processing in Python, 2017. URL `https://spacy.io`.

[128] Tarun Yadav and Arvind Mallari Rao. Technical Aspects of Cyber Kill Chain. In Jemal H Abawajy, Sougata Mukherjea, Sabu M Thampi, and Antonio Ruiz-Mart´`insertBlock`nez, editors, *Security in Computing and Communications*, volume 536, pages 438–452, Cham, 2015. Springer International Pub-

lishing. ISBN 978-3-319-22915-7. doi: 10.1007/978-3-319-22915-7_40. URL http://link.springer.com/10.1007/978-3-319-22915-7{_}40.

[129] James Hamilton, Manuel Gonzalez Berges, Jean-Charles Tournier, and Brad Schofield. SCADA Statistics monitoring using the elastic stack (Elasticsearch, Logstash, Kibana). In *Proc. of International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS'17), Barcelona, Spain, 8-13 October 2017*, pages 451–455, Geneva, Switzerland, 2018. JACoW. ISBN 9783954501939. doi: https://doi.org/10.18429/JACoW-ICALEPCS2017-TUPHA034. URL http://jacow.org/icalepcs2017/papers/tupha034.pdf.