High-Performance Unsupervised Anomaly Detection for Cyber-Physical System Networks

Peter Schneider
Fraunhofer Institute for Applied and Integrated Security
Garching
peter.schneider@aisec.fraunhofer.de

Konstantin Böttinger
Fraunhofer Institute for Applied and Integrated Security
Garching
konstantin.boettinger@aisec.fraunhofer.de

ABSTRACT

While the ever-increasing connectivity of cyber-physical systems enlarges their attack surface, existing anomaly detection frameworks often do not incorporate the rising heterogeneity of involved systems. Existing frameworks focus on a single fieldbus protocol or require more detailed knowledge of the cyber-physical system 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 on creating an efficient framework which can also handle the increased amount of communication in cyber-physical systems.

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%.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; *Network security*; • **Computer systems organization → Embedded and cyber-physical systems**;

KEYWORDS

anomaly detection, cyber-physical systems, deep neural network, network traffic

ACM Reference Format:
Peter Schneider and Konstantin Böttinger. 2018. High-Performance Unsupervised Anomaly Detection for Cyber-Physical System Networks. In *Workshop on Cyber-Physical Systems Security & Privacy (CPS-SPC '18), October 19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3264888.3264890

1 INTRODUCTION

Ongoing digitization and automation are driving factors in today's industry and manufacturing innovations. The integration of sensors and actuators with state-of-the-art data analytics and control algorithms leads to the rise of cyber-physical systems (CPS). These are characterized by a closed feedback loop between the physical world and their digital control systems. Similar systems have long since existed for plant control and internal industrial automation. Also referred to as Supervisory Control and Data Acquisition (SCADA), these systems were designed to operate in isolated environments yielding a secure configuration to the greatest possible extent.

Today, these environments are not isolated anymore. Instead, even small sensors and actuators get connected to other internal networks, the Internet, and to external cloud services. While the amount of such connected cyber-physical systems increases, they face new and evolving information security threats [4]. Major risks originate from attacks targeted at manipulating the underlying physical process of the system (e.g. as in the Stuxnet attack [22]).

One possible method to detect these new and still unknown threats is searching for anomalies in the network. Although there already exist frameworks and methods for network-based anomaly detection for classical IT, they cannot directly be applied to network traffic in industrial settings. Current network anomaly detection systems are based on metadata analysis, i.e. they focus on the headers of network packets. Transferring this approach to industrial network protocols is problematic because of two reasons. First, the differing ratio between average packet header and packet payload length yields more data to process for industrial applications. While in classical business IT the payload of a network packet can reach hundreds of bytes, the payloads in industrial traffic are much shorter resulting in more packets to process in a similar amount of time. Second, neglecting the payload of network packets falls short of making the detection of application logic manipulation attacks possible. These attacks, also referred to as programming logic controller (PLC) exploitation, manipulate the behavior of one system so that reactions of its communication partners are steered into the desired direction [4]. As the attack is limited to the application logic, the original communication pattern, i.e. the packet headers, do not differ from normal system behavior. Langner [22] revealed that such an attack step was also part of the Stuxnet attack.

Another challenge involved is the use of uncommon and proprietary protocols. As each manufacturer of devices may choose out of a variety of industrial network protocols, inspection of network packets requires a thorough understanding of the used protocols and compatible processing strategies for each of them.

To address these challenges, 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.

2 RELATED WORK

Developments in recent years have shown the attack and damage potential involved with the ever-increasing connectivity of SCADA and industrial Internet of Things (IoT) applications. Our contribution is related to machine-learning based anomaly detection in cyber-physical systems as addressed in the following.

2.1 Cyber-physical System Security

Some recent attacks have been analyzed in detail in the past [6, 22]. Antón et al. [4] give a summary of current exploits and trends in attacks on CPS. The works of [6, 10, 22] give insights into protection goals for industrial IoT. The reports show that attackers must be assumed to be very knowledgeable about the attacked systems. Attackers effectively manipulated application logic of SCADA systems which developers believe to be too specialized to be understood by someone not involved in its creation. As application logic is not part of network packet headers, anomaly detection systems based on metadata analysis may not see these manipulations. A study by [30] showed that under 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.

Further studies showed that CPSs have specific properties differentiating them from classical business IT. Fernandes et al. [13] claim that the well-defined behavior of a CPS eases the learning of system models for anomaly detection.

In our approach we take advantage of this well-defined behavior to train a stacked denoising autoencoder completely unsupervised.

2.2 Machine-Learning based anomaly detection

While applying machine learning to network anomaly detection is not a new idea, current methods do not take advantage of the special properties of CPSs mentioned before. Anomaly detection using metadata or telemetry analysis for specific setups has already been explored [3, 9, 27, 32].

Turner et al. [30] and Pasqualetti et al. [26] pointed out that neglecting the actual content of the network packets might result in inferior attack detection. Thus, [2, 16, 21] were able to demonstrate successful manipulations of process logic which are undetectable by metadata-based anomaly detection. To detect such command or data injection attacks, Beaver et al. [5] tested machine-learning based methods in the context of SCADA communication. In contrast to our approach, they extracted protocol-specific fields of the network packet payloads to analyze the traffic.

An application of an ensemble of autoencoders for network anomaly detection was presented very recently with the `Kitsune` framework [25]. 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.

Apart from methods analyzing the network traffic, there are scenario-adapted anomaly detection approaches which require the modeling and understanding of the underlying CPS [18]. Similarly, Potluri et al. [28] 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 which extracts the required information automatically from network packets captured in the CPS.

3 CONCEPT

Prior to 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 master and which as slave, do 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 [30]. Most of the involved machines use proprietary or custom-developed protocols making deep packet inspection an unsolvable 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 [23]. Hence, we can assume that if an attacker is able to degrade the production quality of the system, he is also able to correctly implement the network protocol logic. Given these considerations, anomaly detection in this setting should focus on detecting changes in process-relevant control parameters, control timings, and relevant software behavior. Additionally, remote attacks disturbing the availability of the system are likely to be seen.

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 deep knowledge and understanding of the relevant cyber-physical process, it is applicable to multiple threat scenarios, protocols, and environments without scenario-specific adaptations.

3.1 Anomaly Detection as a Classification Problem

Applying machine-learning to anomaly detection requires one to properly model the system 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 [20].

3.1.1 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 tradeoff 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 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

$$p^* = \begin{cases} (p_1, \ldots, p_n, \underbrace{0, \ldots, 0}_{MIS-n}) & \text{if } n < MIS \\ (p_1, \ldots, p_{MIS}) & \text{if } n \geq MIS \end{cases} \tag{1}$$

3.2 Deep Autoencoders for unified feature learning and classification

As described in Section 3.1, the network packets are trimmed or padded to a specified input size. This enables fast pre-processing 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.

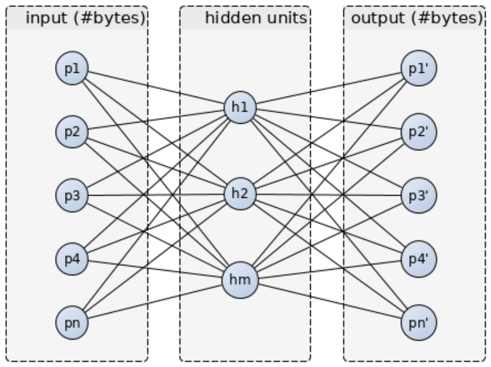

Figure 1: Schematic drawing of one autoencoder layer.

An autoencoder is a fully-connected neural network trained for replicating the input values as its output [7]. 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 1. When denoting the values in each node of a layer as a vector, the values of the hidden layer can be written as

$$h = a(W_e p + b_e) \tag{2}$$

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

$$y = a(W_d h + b_d) \tag{3}$$

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

$$se = |p - y|^2 \tag{4}$$

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

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 pre-processing step is not needed

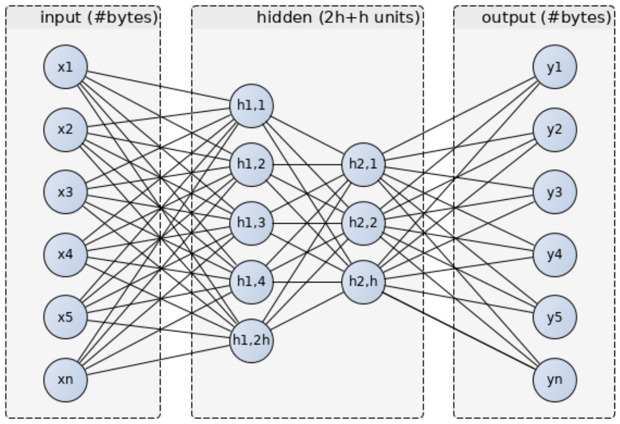

Figure 2: SDA architecture for anomaly detection on network packets.

in this case. The introduction of stacked autoencoder layers is also referred to as deep neural networks [19]. 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 2.

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

$$rmse(p^*) = \sqrt{(p^* - f(p^*))^2} \tag{5}$$

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

$$p \text{ is } \begin{cases} \text{negative} & \text{if } t_{low} < rmse(p^*) < t_{up} \\ \text{positive} & else \end{cases} \tag{6}$$

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 SDA and the original vector p^*. This is a common approach for avoiding overfitting in stacked denoising autoencoders [31].

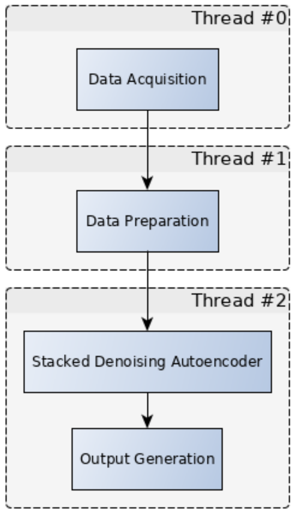

Figure 3: Parallelized Processing using Pipelining.

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.

3.3 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 processing pipeline for big data analysis is the map-reduce approach [12]. 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 3 (Thread #0). Second, the padding and length cutoff 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 (GPU) [29], 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 the time and yields more stable results.

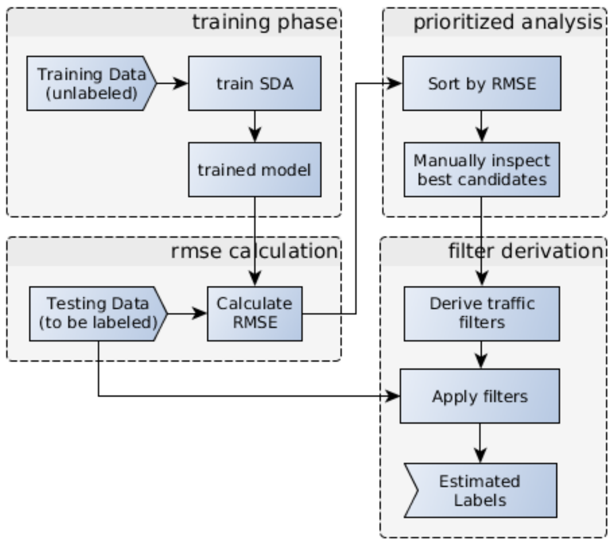

Figure 4: Semi-automatic estimation of traffic labels.

Hence, we later use the *rmse* of a batch of packets *B* defined as

$$rmse(B) = \frac{1}{|B|} \sum_{p \in B} \sqrt{(f(p^*) - p^*)^2} \tag{7}$$

3.4 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 [15]. 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 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 4 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 which 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 1 to 7 (*rmse* calculation). Sorting the batches based on their respective reconstruction 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 are able to manually identify their class as being normal or anomalous traffic. 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 4.1).

As only 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 4.1) 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.

4 IMPLEMENTATION AND EVALUATION

We implemented the proposed framework in *python* using the *tensorflow* framework [1] for processing. As we focused on implementing a real-time capable processing pipeline, we evaluated three packet acquisition methods. Although packet parsing negatively influences the processing time for each network packet, it is included in most off-the-shelf tools for network forensics like python's `scapy` [8] module or the wireshark [11] wrapper `pyshark` [17]. Comparing processing times for different amounts of packets (c.f. Table 1 and Figure 5) 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 [14] 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 JSON structures, they introduce an overhead for each processed packet. The `pcap` module, instead, directly forwards a sequence of

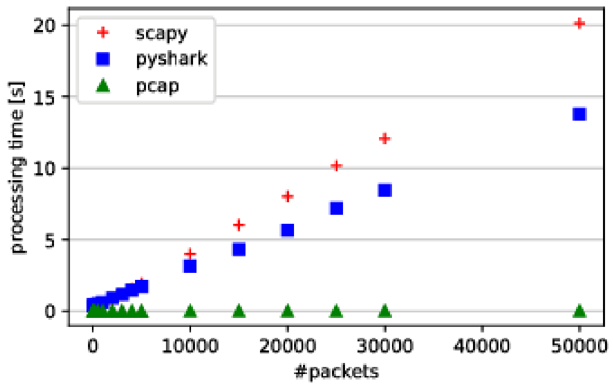

Figure 5: Performance of different packet acquisition libraries.

# 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 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 a i7-4600U linux machine @$3.3GHz$ and $8GB$ RAM.

bytes. In our evaluation datasets, using the packet parsing-based methods makes real-time processing almost impossible. The Secure Water Treatment (SWaT) S3 dataset 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 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.

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 which needs to acquire and forward the packets sequentially and in time.

4.1 Deriving qualitative measures

Considering the definitions of *positive* and *negative* given in Section 3.1, 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 a 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, instead, 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

$$precision = \frac{tp}{tp + fp} \tag{8}$$

$$recall = \frac{tp}{tp + fn} \tag{9}$$

$$f1 = 2\frac{precision \cdot recall}{precision + recall} \tag{10}$$

4.2 Modbus data

In a first run, 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 [24]. 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 which does not contain any attacks. For evaluating the detection approach, we tested all other traces which include attacks. The summary of quality measures in Table 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 6 show the $rmse$ for batches of 200 packets in the `run1_3rtu_2s` trace, i.e. the $rmse$ on the training data,

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 2: Quality measures for the Modbus dataset according to the labels by [24] when applying thresholds $t_{up} = 2000, t_{low} = 500$.

despite of a scaling factor. Figure 7 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 [24] 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 packets starting directly after the previous cross up to the cross itself have the same error value.

The *rmse* during training (c.f. Figure 6) 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 7, it can be seen that the *rmse* rises, once the attacks started. The first data point has still 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 8 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 reliably detect anomalies. Hence, the corresponding line in Table 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.

In Figure 9 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 [24]. 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.

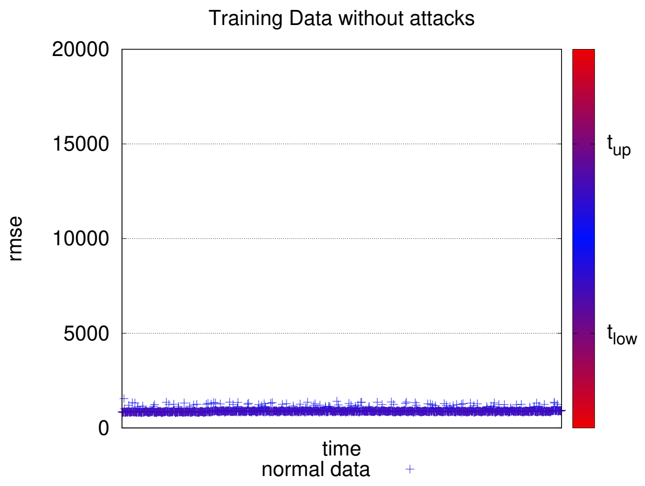

Figure 6: RMSE on the `run1_3rtu_2s` trace used for training on the Modbus dataset by [24].

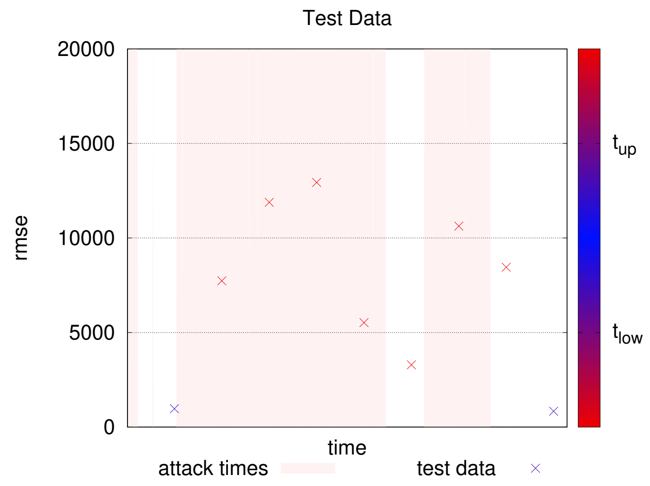

Figure 7: RMSE on the `exploit_ms08_netapi_modbus_6RTU_with_operate` trace.

4.3 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 300*GB* of data collected during several days in the Secure Water Treatment (SWaT) S3 event. 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) [15]. The data is split into two parts, one which does not contain any attacks and another one which does include attacks. However, that data 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 6*GB*. For the training phase, we use the `packet_00019_20170614105105.cap`

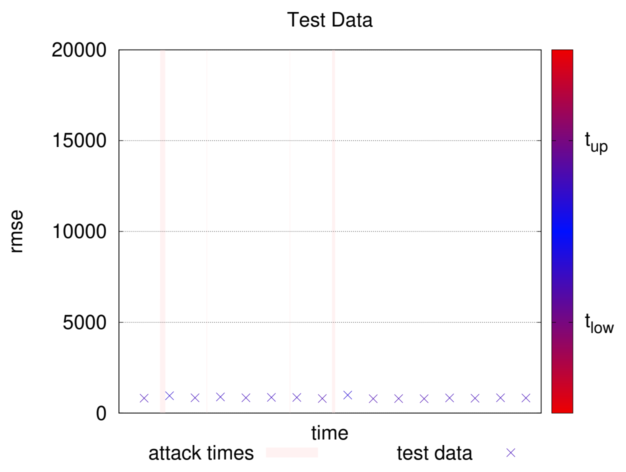

Figure 8: RMSE on the `moving_two_files_modbus_6RTU` trace.

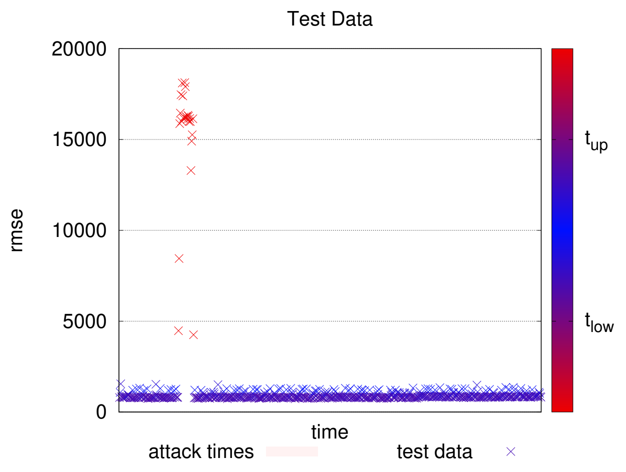

Figure 9: RMSE on the `run1_6rtu` trace.

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 individual trace itself is at least one-hour long. Figure 10 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 4.1. 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 104*GB*. 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 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 corresponding *rmse* values for each packet batch. As shown in Figure 11, in contrast to the training and validation datasets (cf. Figure 10), the *rmse* occasionally exceeds the value of 1500 which had been an upper limit throughout the whole validation set.

4.3.1 Estimating Labels for Unlabeled Datasets.

To derive labels for the second part of the SWaT dataset, we apply our labeling approach outlined in Section 3.4. This enables us to identify main root causes for 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 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
```

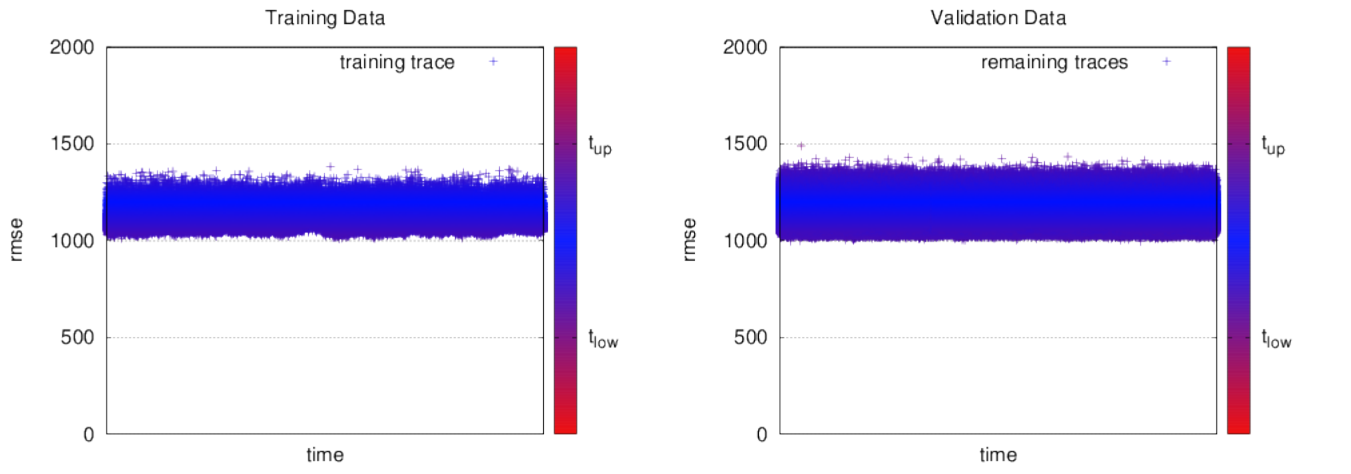

Figure 10: Root mean squared error during training and on validation data from SWaT.

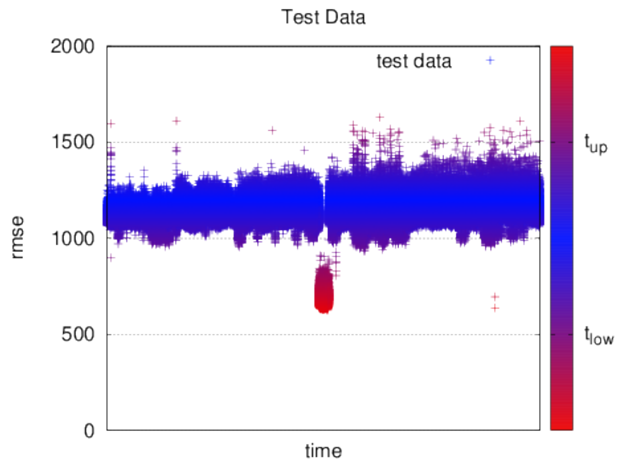

Figure 11: RMSE on the s3171 trace.

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 for anomalous actions going on. In cyber-physical systems, 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 a labeling for a portion of the network traffic. These labels can then be used for the calculation of quality measures.

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 3 and 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 which 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 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

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

Table 3: Anomaly detection performance in problematic scenarios.

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

Table 4: Anomaly detection performance in well-working scenarios.

fewer batches. Remembering the findings from the Modbus dataset (cf. Figure 8 and explanation), this might also be a consequence of our batching method.

Considering the results of the more frequent anomalies shown in Table 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 11. 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 normal 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 3.2 can also be used to detect such shifts in network behavior.

4.3.2 Comparison against naive classifiers. Given the Equations 1 to 7, training a SDA for anomaly detection may lead to a naive classifier. As we padd 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

$$e = |avg_p - avg_{B_i}| \tag{11}$$

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 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 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 and Table 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

Line		tcpreset			synflood		
		p	n	∑	p	n	∑
1	t	0	976754	976754	0	976772	976772
2	f	1638	21588	23226	1638	21570	23208
3	∑	1638	998342	999980	1638	998342	999980
4	precision		0.00%			0.00%	
5	recall		0.00%			0.00%	
6	f1		0.00%			0.00%	

Table 5: Anomaly detection performance using a naive approach.

training to detect these attacks. For SYN floods, one possible approach would be to mask packets going to ports, which are unused during training, of each machine. 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 a long-term applicability of a trained model as long as the general system and network behavior does not change.

5 CONCLUSION

We present a method and framework for efficient and effective anomaly detection in cyber-physical systems' 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 [24] as well as in the Secure Water Treatment dataset [15] can be found. The ability to construct anomaly detection systems independent of the analyzed protocols enables broad applications in the currently evolving industrial internet-of-things and cyber-physical systems. 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 very restricted.

PROJECT FUNDING

The presented work is part of the German national security reference project IUNO (https://www.iuno-projekt.de). The project is funded by the BMBF and aims to provide building-blocks for security in the emerging field of Industrie 4.0.

ACKNOWLEDGMENTS

We thank iTrust, Centre for Research in Cyber Security, Singapore University of Technology and Design for providing the SWaT dataset and, thereby, supporting our work.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*, Vol. 16. 265–283.
[2] Ali Abbasi and Majid Hashemi. 2016. Ghost in the PLC Designing an Undetectable Programmable Logic Controller Rootkit via Pin Control Attack. *Black Hat Europe* (2016), 1–35.
[3] Abdulmohsen Almalawi, Adil Fahad, Zahir Tari, Abdullah Alamri, Rayed AlGhamdi, and Albert Y Zomaya. 2016. An efficient data-driven clustering technique to detect attacks in SCADA systems. *IEEE Transactions on Information Forensics and Security* 11, 5 (2016), 893–906.
[4] Simon Duque Antón, Daniel Fraunholz, Christoph Lipps, Frederic Pohl, Marc Zimmermann, and Hans D Schotten. 2017. Two decades of SCADA exploitation: A brief history. In *Application, Information and Network Security (AINS), 2017 IEEE Conference on.* IEEE, 98–104.
[5] Justin M Beaver, Raymond C Borges-Hink, and Mark A Buckner. 2013. An evaluation of machine learning methods to detect malicious SCADA communications. In *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*, Vol. 2. IEEE, 54–59.
[6] B Bencsáth, G Pék, L Buttyán, and M Felegyhazi. 2012. sKyWIper (aka Flame aka Flamer): A complex malware for targeted attacks. *CrySyS Lab Technical Report, No. CTR-2012-05-31* (2012).
[7] Yoshua Bengio et al. 2009. Learning deep architectures for AI. *Foundations and trends® in Machine Learning* 2, 1 (2009), 1–127.
[8] Philippe Biondi. 2011. Scapy. https://scapy.net/.
[9] Marco Caselli, Emmanuele Zambon, and Frank Kargl. 2015. Sequence-aware intrusion detection in industrial control systems. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security.* ACM, 13–24.
[10] Thomas M Chen. 2010. Stuxnet, the real start of cyber warfare?[Editor's Note]. *Network, IEEE* 24, 6 (2010), 2–3.
[11] Gerald Combs. 2006. Wireshark. *http://www.wireshark.org* (2006).
[12] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters [J]. *Commun. ACM* 1 (2008), 107–113.
[13] Earlence Fernandes, Amir Rahmati, Kevin Eykholt, and Atul Prakash. 2017. Internet of Things Security Research: A Rehash of Old Ideas or New Intellectual Challenges? *arXiv preprint arXiv:1705.08522* (2017).
[14] Arturo Filastò. 2004. pypcap. https://github.com/pynetwork/pypcap.
[15] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. 2016. A dataset to support research in the design of secure water treatment systems. In *International Conference on Critical Information Infrastructures Security.* Springer, 88–99.
[16] Naman Govil, Anand Agrawal, and Nils Ole Tippenhauer. 2017. On Ladder Logic Bombs in Industrial Control Systems. *arXiv preprint arXiv:1702.05241* (2017).
[17] Dor Green. 2013. pyshark. http://kiminewt.github.io/pyshark/.

[18] Piroska Haller and Béla Genge. 2016. Using Sensitivity Analysis and Cross-Association for the Design of Intrusion Detection Systems in Industrial Cyber-Physical Systems. *IEEE*, Article DOI 10.1109/ACCESS.2017.2703906 (2016).

[19] Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science* 313, 5786 (2006), 504–507.

[20] Charles Hornig. 1984. *RFC 894: Standard for the transmission of IP datagrams over Ethernet networks.* Technical Report. RFC, IETF, April.

[21] Amit Kleinmann, Ori Amichay, Avishai Wool, David Tenenbaum, Ofer Bar, and Leonid Lev. 2017. Stealthy Deception Attacks Against SCADA Systems. *arXiv preprint arXiv:1706.09303* (2017).

[22] Ralph Langner. 2013. To kill a centrifuge: A technical analysis of what stuxnet's creators tried to achieve. *Online: http://www. langner. com/en/wp-content/uploads/2013/11/To-kill-a-centrifuge. pdf* (2013).

[23] Robert M. Lee, Michael J. Assante, and Tim Conway. 2016. Analysis of the cyber attack on the Ukrainian power grid. *Electricity Information Sharing and Analysis Center (E-ISAC)* (2016).

[24] Antoine Lemay and José M Fernandez. 2016. Providing SCADA network data sets for intrusion detection research. In *9th Workshop on Cyber Security Experimentation and Test (CSET 16)*. USENIX Association.

[25] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. *Network and Distributed Systems Security Symposium (NDSS)* (2018).

[26] Fabio Pasqualetti, Florian Dörfler, and Francesco Bullo. 2013. Attack Detection and Identification in Cyber-Physical Systems. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL* 58, 11 (November 2013), 2715–2729.

[27] Stanislav Ponomarev and Travis Atkison. 2016. Industrial Control System Network Intrusion Detection by Telemetry Analysis. *IEEE Transactions on Dependable and Secure Computing* 13, 2 (2016), 252–260.

[28] Sasanka Potluri, Christian Diedrich, and Girish Kumar Reddy Sangala. 2017. Identifying false data injection attacks in industrial control systems using artificial neural networks. In *Emerging Technologies and Factory Automation (ETFA), 2017 22nd IEEE International Conference on.* IEEE, 1–8.

[29] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. 2016. Benchmarking state-of-the-art deep learning software tools. In *Cloud Computing and Big Data (CCBD), 2016 7th International Conference on.* IEEE, 99–104.

[30] Hamilton Turner, Jules White, Jaime A Camelio, Christopher Williams, Brandon Amos, and Robert Parker. 2015. Bad parts: Are our manufacturing systems at risk of silent cyberattacks? *IEEE Security & Privacy* 13, 3 (2015), 40–47.

[31] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*. ACM, 1096–1103.

[32] Jiexin Zhang, Shaoduo Gan, Xiaoxue Liu, and Peidong Zhu. 2016. Intrusion detection in SCADA systems by traffic periodicity and telemetry analysis. In *2016 IEEE Symposium on Computers and Communication (ISCC)*. IEEE, 318–325.