Poisoning Attacks on Cyber Attack Detectors for Industrial Control Systems

Moshe Kravchik
Ben-Gurion University of the Negev
moshekr@post.bgu.ac.il

Battista Biggio
University of Cagliari
battista.biggio@unica.it

Asaf Shabtai
Ben-Gurion University of the Negev
shabtaia@bgu.ac.il

ABSTRACT

Recently, neural network (NN)-based methods, including autoencoders, have been proposed for the detection of cyber attacks targeting industrial control systems (ICSs). Such detectors are often retrained, using data collected during system operation, to cope with the natural evolution (i.e., concept drift) of the monitored signals. However, by exploiting this mechanism, an attacker can fake the signals provided by corrupted sensors at training time and poison the learning process of the detector such that cyber attacks go undetected at test time. With this research, we are the first to demonstrate such poisoning attacks on ICS cyber attack online NN detectors. We propose two distinct attack algorithms, namely, interpolation- and back-gradient based poisoning, and demonstrate their effectiveness on both synthetic and real-world ICS data. We also discuss and analyze some potential mitigation strategies.

KEYWORDS

Anomaly detection; industrial control systems; autoencoders; adversarial machine learning; poisoning attacks; adversarial robustness.

ACM Reference Format:
Moshe Kravchik, Battista Biggio, and Asaf Shabtai. 2021. Poisoning Attacks on Cyber Attack Detectors for Industrial Control Systems. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3412841.3441892

1 INTRODUCTION

Neural network-based anomaly and attack detection methods have attracted significant attention in recent years. The ability of neural networks to accurately model complex multivariate data has contributed to their use as detectors in various areas ranging from medical diagnostics to malware detection and cyber-physical system monitoring. The effectiveness of a detection system depends heavily on its robustness to attacks that target the detection system itself. In the context of NNs, such attacks are known as adversarial learning attacks. Adversarial attacks have been a major focus of the NN research community, primarily in the image classification [26, 28], malware detection [24, 28], and network intrusion detection [18, 25] domains.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SAC '21, March 22–26, 2021, Virtual Event, Republic of Korea
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8104-8/21/03…$15.00
https://doi.org/10.1145/3412841.3441892

This research focuses on NN-based anomaly and cyber attack detectors in industrial control systems (ICSs), which are a subclass of cyber-physical systems (CPSs). ICSs are central to many important areas of industry, energy production, and critical infrastructure. The security and safety of ICSs are therefore of the utmost importance. Recent adoption of remote connectivity and common information technology (IT) stack by ICSs exposed them to cyber threats. Despite sharing threat vectors with typical IT systems, ICSs differ from them in several key aspects. First, ICSs are very diverse because each ICS is tailored to the specific process. Second, many ICS threat actors are state-sponsored groups creating dedicated zero-day attacks targeting specific facility or family of facilities [31]. Yet, defending ICS is challenging due to the need to support legacy protocols built without modern security features, limited endpoints computation capabilities, and hard real-time constraints. Hence, traditional IT cyber defence tools are less effective in the ICS context. Consequently, using NN-based techniques for monitoring and attacks detection in ICS became a major trend recently [2–4, 8, 12–16, 29, 30, 32, 33]. Despite their increasing popularity, the resilience of such detectors to adversarial attacks has received little attention.

Adversarial attacks on NNs can be broadly divided into evasion and poisoning attacks. Both kinds of attacks use maliciously crafted data to achieve their goals. **Evasion** attacks aim to craft test data samples that will evade detection while still producing the desired adversarial effect (e.g., service disruption). Recently, three studies examining adversarial attacks on NN-based ICS detectors were published, all of which were dedicated to evasion attacks ([2, 15, 33]) performed using different threat models. As demonstrated in [15], conducting an attack with a significant physical impact on the controlled process while evading a NN detector might be very hard for an attacker controlling only a subset of sensors. In such cases, the attacker might attempt **poisoning** attacks, which have not been studied in the ICS context yet. **Poisoning** attacks attempt to introduce adversarial data when the model is being trained. This data influences the model in such a way that the attack remains undetected at test time. The importance of poisoning attack research increases in light of the popularity of monitoring systems' online training mode. With online training, the model is periodically trained with new data collected from the protected system to accommodate for the concept drift. It was demonstrated in [33] on two public datasets collected from the same testbed at two different times, that over time such drift can be significant enough to render the detector trained on the old data useless. The problem was addressed by detector's retraining with the new data. Another ICS online trained predictor is described in [21], where it is used to predict mass flow in an industrial boiler and is retrained to address the concept drift caused by a nonstandardaized feeding process. Such online retraining provides the adversary with the

opportunity to poison the model; this underscores the need to study poisoning attacks on ICS anomaly and attack detectors.

Problem statement. The threat model assumed in our research considers an adversary whose goal is to change a physical-level process of the targeted ICS, which includes an online-trained anomaly detector. The adversary considered has gained control of a sensor or a number (subset) of sensors and is able to falsify the sensors' readings. Spoofed sensory data can cause the controller of the ICS to issue commands driving the system to a specific state desired by the attacker. The attacker needs to change the controlled sensor's data gradually in the direction that would lead the detector to accept the planned attack as normal behavior. The changes introduced should neither be detected as anomalous by the detector nor cause the detector to detect the normal data as anomalous, thus increasing the problem's difficulty.

Our study aims to answer the following research questions: (1) What algorithms can be used to effectively generate poisoning input for a NN-based ICS anomaly detector operating in online training mode? (2) How robust are the detectors proposed in [2, 15, 29] to such poisoning attacks? and (3) How can such attacks be mitigated?

The contributions of this paper are as follows: *i)* To the best of our knowledge, we present the first study of poisoning attacks on online-trained NN-based detectors for multivariate time series. *ii)* We propose two algorithms for the generation of poisoning samples in such settings: an interpolation-based algorithm and a back-gradient optimization-based algorithm. *iii)* We implement and validate both algorithms on synthetic data and evaluate the influence of various test parameters on the poisoning abilities of the algorithms. *iv)* We apply the algorithms to an autoencoder-based anomaly detector for real-world ICS data and study the detector's robustness to poisoning attacks. *v)* We propose a number of mitigation techniques against poisoning attacks. *vi)* The implementation of both algorithms and the evaluation test code are open source and freely available.[1]

2 INDUSTRIAL CONTROL SYSTEMS

The typical architecture of an ICS is illustrated in Figure 1. ICSs consist of network-connected computers that monitor and control physical processes. These computers obtain feedback about the monitored process from sensors and can influence the process using actuators, such as pumps, engines, and valves. Typically, the sensors and actuators are connected to a local computing element, a programmable logic controller (PLC), which is a real-time specialized computer that runs a control loop supervising the physical process. The PLCs, sensors, and actuators form a remote segment of the ICS network. The other ICS components reside in a different network segment, i.e., the control segment, which typically includes a supervisory control and data acquisition (SCADA) workstation, a human-machine interface (HMI) machine, and a historian server. The SCADA computer runs the software responsible for programming and controlling the PLC. The HMI receives and displays the current state of the controlled process, and the historian keeps a record of all of the sensory and state data collected from the PLC.

Figure 1 also highlights that ICSs are typically attacked either at the sensor (1) or at the PLC (2) level in the remote segment [5, 6].

[1] https://github.com/mkravchik/poisoning-ics-ad

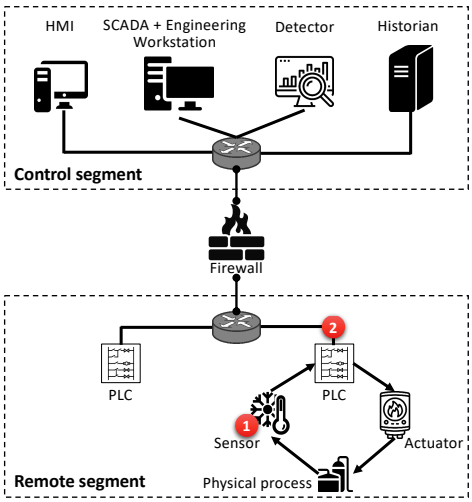

Figure 1: ICS architecture and attack entry points. ICSs can be attacked either at the sensor (1) or at the PLC (2) level, corrupting their outputs to mislead the actuators.

Cyber Attack Detection. The cyber attack detector (or intrusion detection system - IDS) for ICSs is typically located in the control segment. Among the various approaches used to build such detectors, in this work we focus on IDSs that model the physical behavior of the system [5, 6, 11, 19]. These IDSs are thus expected to detect anomalies when the observed physical system state deviates significantly from the expected behavior predicted by their underlying model. Our research focuses on detectors that use NNs to model the monitored process. This approach has recently become very popular due to the ability of NNs to model complex multivariate systems with nonlinear dependencies among the variables [2, 14–16, 29, 30]. Various NN architectures have been used to this end, including convolutional NNs, recurrent NNs, feedforward NNs, and autoencoders [9]. While our attacks are **agnostic to the detector's architecture and can be applied to any kind of NN**, we chose to evaluate them against autoencoders in this paper due to their increasing popularity [2, 15, 29].

Threat Model. To evaluate robustness of such detectors to poisoning attacks, we consider a malicious sensor threat model widely studied in the wireless sensor network domain [22, 27], in a related ICS research [10], and used in a real-world attack-defense exercise for smart grids [23]. It is illustrated in Figure 1, with the location of the attacker marked as (1). Under this model, the attacker possesses knowledge of the historical values measured by the sensors and can spoof arbitrary values of the sensors' readings; however, both the PLC and detector see the same spoofed values. This model is highly relevant to the ICS domain. Consider an ICS with sensors distributed over a large area, which send their data to a PLC residing at a **physically protected and monitored location**. In this setup, the attacker can replace the original sensor with a malicious one, reprogram the sensor, influence the sensor externally, or just send false data to the PLC over the cable/wireless connection, but the attacker cannot penetrate the physically protected PLC-to-SCADA

network. Even though redundant sensors might be deployed, they protect against faults but not targeted attacks. The adversary can attack all related sensors, as they have the same location, protection, and, commonly, the same supply chain. We argue that this setup is much more realistic than one in which the attacker controls both point (1) and point (2) in Figure 1, i.e., the internal network of the remote segment or even the network of the control segment, as considered in [2, 33]. Moreover, the setting considered in this study presents more constraints and challenges to the attacker, who must achieve his/her goals with a single point of data manipulation. Finally, as this is the first work to demonstrate that poisoning cyber attack detectors for ICSs is possible, we assume a white-box attack scenario in which the attacker has full knowledge of the detector (e.g., learned parameters and retraining frequency). As ICSs are commonly attacked by state-sponsored actors, assuming the most knowledgeable adversary is not unrealistic. Related researches [1, 4, 32] use the white-box threat model as well. If the retraining schedule is not known, the attacker can still apply the same algorithms to calculate the poisoning samples, estimate the maximal retraining period, and increase the intervals between the poison injections to become larger than this estimation.

3 POISONING ONLINE ATTACK DETECTORS

We describe here our poisoning attack against learning-based cyber attack detectors for ICSs. The attacker's ultimate goal is to allow a specific attack at test time to stay undetected, despite involving significant changes to the values measured by one or more sensors. For example, the attacker might aim to report a very low water level in a tank, while in reality the tank is full, thus causing it to overflow. Without poisoning, the detector would raise an alert upon encountering such spoofed water-level value, as it deviates significantly from the level that is normally observed. With poisoning, instead, we will show that the attacker can successfully compromise the learning process of the detector to allow specific attacks at test time, without substantially affecting other normal system operations.

To poison the detector, the attacker exploits the fact that it is periodically retrained by using newly-collected data from the monitored system. Within this scenario, we assume that the measured sensor values of the monitored system are added to the training data, and the attacker knows when to inject his/her poisoning samples to be used for retraining. If the detector's training schedule is unknown to the attacker, he/she will have to inject poisoning samples multiple times. We also assume that only data that does not trigger alerts will be used for retraining, i.e., only *gradual* changes are admitted. This makes our poisoning attack more challenging, as the poisoning samples themselves will also have to stay undetected.

Notation. We denote with $\mathbf{y}_t \in \mathbb{R}^d$ the d-dimensional vector consisting of the sensor measurements and actuator states observed by the PLC at time t, and with $\mathbf{Y}_{t,L} = (\mathbf{y}_{t-L}, \ldots, \mathbf{y}_t) \in \mathbb{R}^{d \times L}$ a sequence of such signals from time $t-L$ to time t, being L the *sequence length*. We consider cyber attack detectors based on undercomplete autoencoders [9] that predict such values at each point in time t as:

$$\hat{\mathbf{Y}}_{t,L} = f(\mathbf{Y}_{t,L}, \mathbf{w}), \tag{1}$$

where $\mathbf{w}$ are the model parameters. They are learned during training by minimizing a loss function $\mathcal{L}(\mathcal{D}_{\text{tr}}, w)$ on the training data $\mathcal{D}_{\text{tr}} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$, with $N >> L$. The mean squared error (considering each predicted sequence separately) is normally used to this end:

$$\mathcal{L}(\mathcal{D}_{\text{tr}}, w) = \sum_{t=L+1}^{N} \sum_{i=t-L}^{t} \|\hat{\mathbf{y}}_i - \mathbf{y}_i\|_2^2 . \tag{2}$$

The detection mechanism works as follows. A test input sequence $\mathbf{Y}_{t,W} = (\mathbf{y}_{t-W}, \ldots, \mathbf{y}_t)$ of length W at time t (possibly with $W < L$ for prompt detection) is compared against the corresponding predictions to compute the residuals $r_k = \|\hat{\mathbf{y}}_k - \mathbf{y}_k\|_\infty$, for $k = t-W, \ldots, t$. Each residual corresponds to the maximum absolute value observed across the d values of the difference $\hat{\mathbf{y}}_k - \mathbf{y}_k$. Then, an attack is detected if the smallest residual in this sequence exceeds the detection threshold τ, i.e., if $g(\mathbf{Y}_{t,W}, \mathbf{w}) > \tau$, where

$$g(\mathbf{Y}_{t,W}, \mathbf{w}) = \min_{k=t-W}^{t} \|\hat{\mathbf{y}}_k - \mathbf{y}_k\|_\infty \tag{3}$$

is the detection function. Note that this detection scheme only detects an attack if all the residuals in the W-sized sequence exceed the threshold. The hyperparameter W has thus to be carefully chosen to successfully detect attacks while avoiding false alarms due to short spikes or quick deviations from the normal behavior.

3.1 Poisoning Attacks

The poisoning attack considered in this work aims to allow a specific attack $\mathbf{Y}^a = (\mathbf{y}_1^a, \ldots, \mathbf{y}_Q^a)$ of length Q to stay undetected at test time, by injecting a carefully-optimized *sequence* of M *poisoning samples* $\mathcal{D}_p = (\mathbf{Y}_1^p, \ldots, \mathbf{Y}_M^p)$ into the training data used by the detector. Note that here each poisoning sample $\mathbf{Y}_k^p = (\mathbf{y}_{k1}^p, \ldots, \mathbf{y}_{kT}^p)$ contains T poisoning points, aiming to capture the natural periodicity of the given physical signals to stay undetected. Our poisoning attack is successful if the detector, trained on $\mathcal{D}_{\text{tr}} \cup \mathcal{D}_p$, does not detect $\mathbf{Y}^a$ at test time, while also not raising false alarms on normal data. The proposed poisoning scenario is summarized in Figure 2.

Our attack can be formulated as a bi-level optimization problem:

$$\mathcal{D}_p^\star \in \arg\min_{\mathcal{D}_p} L(\mathbf{Y}^a, \mathbf{w}^\star), \tag{4}$$

$$\text{s.t.} \quad \mathbf{w}^\star \in \arg\min_{\mathbf{w}} \mathcal{L}(\mathcal{D}_{\text{tr}} \cup \mathcal{D}_p, \mathbf{w}), \tag{5}$$

$$g(\mathcal{D}_p, \mathbf{w}^\star) \preceq \tau, \tag{6}$$

$$g(\mathcal{D}_{\text{val}}, \mathbf{w}^\star) \preceq \tau, \tag{7}$$

where the outer problem in Eq. 4 corresponds to having the attack sample undetected, the inner problem in Eq. 5 amounts to training the autoencoder on the poisoned training data, and the constraints in Eqs. 6-7 respectively require the poisoning samples and normal data (drawn from a validation set) to stay always below the detection threshold τ. Note that the poisoning samples influence the outer objective only indirectly, through the choice of the optimal parameters $\mathbf{w}^\star$ learned from the poisoned training data. The outer objective $L(\mathbf{Y}^a, \mathbf{w})$ is computed as done for the training loss in Eq. 2, i.e., by summing up the mean squared errors computed on sequences of fixed length L: $L(\mathbf{Y}^a, \mathbf{w}) = \sum_{t=L+1}^{Q} \sum_{i=t-L}^{t} \|\hat{\mathbf{y}}_i^a - \mathbf{y}_i^a\|_2^2$.

We propose below two different poisoning algorithms to solve the given bi-level optimization. In both cases, for computational convenience, we greedily optimize one poisoning sample $\mathbf{Y}_k^p$ at a time, and add it to the poisoning set $\mathcal{D}_p$ iteratively.

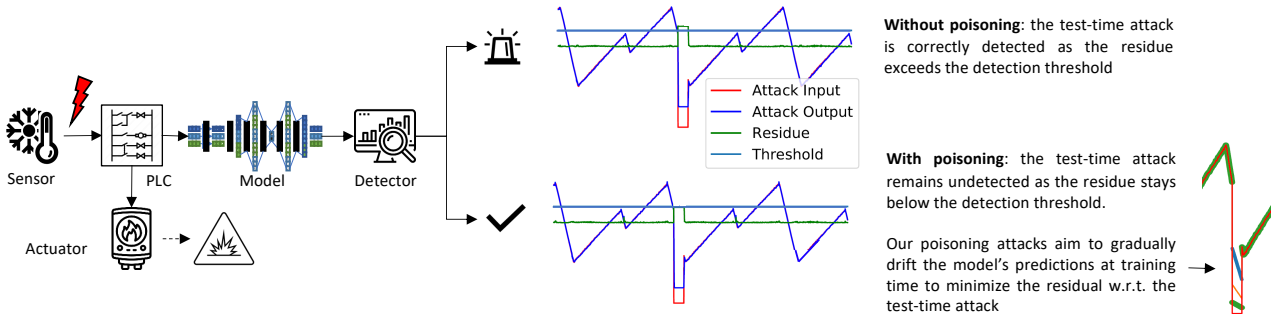

Figure 2: ICS cyber attack detector under a poisoning attack.

3.1.1 Back-gradient optimization attack. One approach for solving Problem 4-7 is to use gradient descent. To keep notation clean, we denote below the poisoning sample $\mathbf{Y}_k^p$ with y_c, and the outer objective with $\mathcal{A}(\mathbf{w}^\star(y_c))$, to clarify that it only depends implicitly on y_c through the selection of the optimal parameters $\mathbf{w}^\star$. Accordingly, the gradient of the outer objective can be computed with the chain rule as:

$$\nabla_{y_c}\mathcal{A} = \frac{\delta \mathbf{w}^\star}{\delta y_c}^T \nabla_w L, \tag{8}$$

where the term $\frac{\delta \mathbf{w}^\star}{\delta y_c}$ captures the change induced in the optimal parameters $\mathbf{w}^\star$ of the autoencoder due to the injection of the poisoning sample y_c into the training set. For some learning algorithms, this term can be computed in closed form by replacing the inner learning problem with its equilibrium conditions [20]. However, this is not practical for deep architectures like autoencoders, as they have an extremely large number of parameters and their equilibrium conditions are typically only loosely satisfied. To tackle these issues, we use back-gradient optimization [17], as suggested in [20]. The core idea of this approach is iterative backwards calculation of both the weights' updates and $\frac{\delta \mathbf{w}^\star}{\delta y_c}$, performed by reversing the learning process and calculating the second gradients in each iteration. We implemented back-gradient optimization for stochastic gradient descent according to Algorithm 1 (based on [20]).

Algorithm 1 Find $\nabla_{y_c}\mathcal{A}$ given trained parameters w_T, learning rate α, attack input y_a, poisoning sample y_c, loss function L, and learner's objective $\mathcal{L}$, using back-gradient descent for T iterations.

1: **function** GETPOISONGRAD(w_T, α, y_a, y_c, L, $\mathcal{L}$)
2: $dy_c \leftarrow 0$
3: $dw \leftarrow \nabla_w L(y_a, w_T)$
4: **for** $t = T$ to 1 **do**
5: $dy_c \leftarrow dy_c - \alpha dw \nabla_{y_c} \nabla_w \mathcal{L}(y_c, w_t)$
6: $dw \leftarrow dw - \alpha dw \nabla_w \nabla_w \mathcal{L}(y_c, w_t)$
7: $gr \leftarrow \nabla_w \mathcal{L}(y_c, w_t)$
8: $w_{t-1} \leftarrow w_t + \alpha gr$
9: **return** dy_c

Algorithm 1 starts with initializing the derivatives of the loss relative to the attack input and the weights of the trained model (lines 2 and 3). Then it iterates for a given number T iterations, rolling back the weight updates made by the training optimizer (lines 7 and 8). In each iteration, the algorithm calculates the second derivatives of the loss relative to the weights and the attack input at the current weights' values (lines 5 and 6) and updates the values maintained for both derivatives. The final value of $\nabla_{y_c}\mathcal{A}$ accumulates the compound influence of the poison input through the weights' updates.

Applying back-gradient optimization to periodic signals. Algorithm 2, which is one of the contributions of this research, was used to apply Algorithm 1 to autoregression learning of periodic signals. Algorithm 2 starts with an empty set of poisoning samples (line 1) and an initial poisoning value. Then it repeatedly uses a *train_test* function to perform the model retraining with the current training and poisoning datasets (line 6). If the target attack input and the clean validation data do not raise alerts, the problem is solved (lines 7-9). Otherwise, the gradient of the poisoning input is calculated using Algorithm 1, normalized, and used to find the next poison value (lines 10-11). The new poison value is tested with the current detector (line 13). If it raises alerts, the value is too large, and the last poison value that did not raise an alert is added to $\mathcal{D}_p$, the adversarial learning rate is decreased, and the last good (capable of being added without raising an alert) poison is used as a base for the calculation in the next iteration (lines 14-17). If the learning rate becomes too low, the algorithm terminates prematurely (lines 18-19). If no alerts were raised, the learning rate is restored to its original value for the next iteration, in order to accelerate the poisoning progress (lines 20-21).

For simplicity, we omitted the adversarial learning rate's (λ) dynamic decay used in the algorithm's implementation from the description of Algorithm 2. With the dynamic decay, λ is decreased if the test error has not decreased, and the iterations are terminated early if $\lambda < 0.00001$. Another implementation optimization not shown in the pseudocode of Algorithm 2 adds *clean* data sequences to $\mathcal{D}_p$ if the detector raises alerts on clean data. If this happens, the model is "over-poisoned" and clean data is added until these alerts disappear. This is done after the calls to *train_test*.

Sliding window prediction poisoning. NNs detectors commonly operate on the multivariate sequences formed by sliding a window of a specified length over the input signal. These sequences are *overlapping*, hence a single time point appears in multiple sequences. As a result, a change to a single time point by an attacker affects the detector's predictions for the multiple sequences that include this point. Moreover, in order for the changed point to remain undetected, its prediction should also be close to its (changed) value based on multiple past input sequences. These self-dependencies spread across time, both forward and backwards, and must be taken

Algorithm 2 Find poisoning sequence set $\mathcal{D}_p$ given $\mathcal{D}_{tr}$, $\mathcal{D}_{val}$, learning rate α, adversarial learning rate λ, attack input y_a, initial poisoning sample y_c^0, loss function L, learner's objective $\mathcal{L}$, and maximum number of iterations M.

$\mathcal{D}_p \leftarrow []$
$decay \leftarrow 0.9$
$eps \leftarrow 0.00001$ ▷ Minimal allowed λ
$orig\lambda \leftarrow \lambda$
for $i = 1$ to M **do**
  $(w_T, alerts) \leftarrow train\_test(\mathcal{D}_{tr}, \mathcal{D}_{val}, y_a, \mathcal{D}_p, y_c^i)$
  if $alerts == 0$ **then**
    D_p += (y_c^i) ▷ Add current poison
    break
  $dy_c \leftarrow getPoisonGrad(w_T, \alpha, y_a, y_c^i, L, \mathcal{L})$
  $y_c^{i+i} \leftarrow y_c^i - \lambda \cdot dy_c / \max(dy_c)$
    ▷ Check that the new poison does not generate alerts
  $(w_T, alerts) \leftarrow train\_test(\mathcal{D}_{tr}, \mathcal{D}_{val}, y_a, \mathcal{D}_p, y_c^{i+i})$
  if $alerts > 0$ **then**
    D_p += (y_c^i) ▷ Add previous poison
    $\lambda \leftarrow decay \cdot \lambda$ ▷ Adjust learning rate
    $y_c^{i+i} \leftarrow y_c^i$ ▷ Revert to last good poison
    if $\lambda <= eps$ **then**
      break ▷ Can't find anymore poisons
  else
    $\lambda \leftarrow orig\lambda$
return $\mathcal{D}_p$

into account when creating the poisoning input, as this input must therefore be much longer than the sequence of points changed during the target attack. However, the model at the attacker's disposal deals only with the short sequences. In order to be able to evaluate the total loss value of the attack for the entire input, we performed the optimization on a **wrapper model ($\mathcal{WM}$)** built around the original trained model.

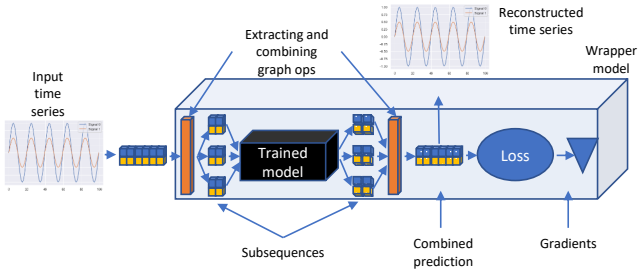

Figure 3: Wrapper model that allows for calculating gradients and optimizing the poisoning samples for arbitrary long input sequence based on an original model predicting short sequences.

The $\mathcal{WM}$ illustrated in Figure 3 extends the trained model's graph to calculate the gradient of the attacker's objective relative to the entire input. The length of this input for periodic signals needs to be at least one period, as the attacker must comply with the signal's normal behavior. Specifically, the $\mathcal{WM}$ prepends the original model with graph operations that divide the long input into

Algorithm 3 Find poisoning sequence set $\mathcal{D}_p$ given $\mathcal{D}_{tr}$, $\mathcal{D}_{val}$, decay rate δ, attack input y_a, and initial poisoning sample y_c^0.

1: $eps \leftarrow 0.0000001$
2: $\mathcal{D}_p \leftarrow []$
3: $rate \leftarrow 1$
4: $step \leftarrow 1$
5: $y_p \leftarrow y_c^0$
6: **while** $\max(|step|) > eps$ **do**
7: $step = rate \cdot (y_a - y_p)/2$
8: $y_c = y_p + step$
9: $(err, alerts) \leftarrow train\_test(\mathcal{D}_{tr}, \mathcal{D}_{val}, y_c, \mathcal{D}_p)$ ▷ Test if current poison raises alert
10: **if** $alerts$ **then**
11: $rate \leftarrow rate \cdot \delta$ ▷ Decrease the rate
12: **else**
13: $y_p \leftarrow y_c$ ▷ Start interpolating from the new point
14: D_p += y_c ▷ Add current poison
15: $rate \leftarrow rate/\delta$ ▷ Increase the rate
16: $(err, alerts) \leftarrow train\_test(\mathcal{D}_{tr}, \mathcal{D}_{val}, y_a, \mathcal{D}_p)$ ▷ Test if the attack raises alert after poisoning
17: **if** $alerts == 0$ **then**
18: **break** ▷ The goal is reached
19: **return** $\mathcal{D}_p$

overlapping subsequences and appends the model with operations that combine the results of individual predictions and calculate the combined output. The $\mathcal{WM}$ allows us to calculate the gradients and optimize the adversarial input for an arbitrary long input sequence.

3.1.2 Interpolation-based attack. In addition to the back-gradient optimization algorithm (Algorithm 1), we propose a much simpler naive interpolation algorithm to identify the poisoning sequence. This algorithm is based on an observation that both the initial poisoning sample and the final attack point are known in advance.

The interpolative Algorithm 3 starts with an empty set of poisoning samples, an initial poisoning sample, and an initial interpolation step (lines 1-5). In each iteration, the algorithm attempts to add a poisoning sample that is an interpolation between the initial point and the final point (lines 7-9). If the new poisoning sample does not raise an alert, it is added to the result set, and the next interpolation between it and the target attack is tested (lines 12-15). Otherwise, the interpolation step is decreased and the interpolation is recalculated (lines 10-11). The algorithm continues until attack is not detected or the interpolation step becomes too small.

4 EXPERIMENTS AND RESULTS

4.1 Evaluation Setup

Anomaly detection model. A simple undercomplete autoencoder (UAE) network was used for the ICS detector under test. We used the network architecture described in [15] for all tests, for both synthetic and real data. The simplest instance of such a detector model is presented in Figure 4.

The autoencoder model includes *tanh* activation in the last layer. This last activation layer constrains the model's output to be between −1 and 1, while the model is trained with data in the range of −0.5 - 0.5. On the one hand, this prevents the attacker from introducing large poison values. On the other hand, it leaves enough space

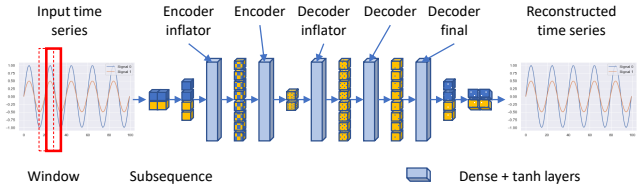

Figure 4: The UAE architecture used in ICS attack detector.

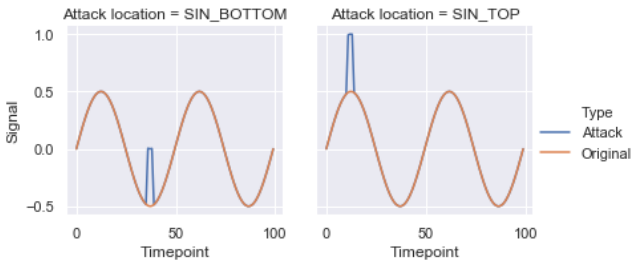

Figure 5: Different attack locations relative to the signal's period. For the SIN_BOTTOM location, the attack stays in the range of the normal signal; for the SIN_TOP it goes beyond this range.

for normal concept drift and for the attacker to execute moderate, under the radar, poisoning.

The detector was implemented in TensorFlow and trained using the gradient descent optimizer for 10-30 epochs until the the mean squared reconstruction error for the input signal decreased to 0.001 and there were no false positives. While we experimented with various amounts of encoder and decoder layers, and multiple inflation factors and input-to-code ratios, these variables mainly influenced the detector's accuracy and not the poisoning results. The results presented in this section are for the tests performed with an inflation factor of two, an input-to-code ratio of two, and a single encoding and decoding layer.

Training process. The model was initially trained using the entire training dataset. For simulating the online training, we performed model retraining starting with the original trained model. After each poisoning iteration, the model was retrained using the newly generated poisoning input as well as the last part of training data (note that only the samples that were not classified as anomalies by the detector were used). There are other possible ways of combining the new and existing training data, e.g., randomly selecting a fixed number of data samples from both. This setup was tested as well and caused a larger number of poisoning samples to be added but did not change the overall findings.

Evaluation metrics. The following metrics were used for the poisoning effectiveness evaluation: (1) the *test time attack magnitude*, measured as the maximal difference between the original and the target spoofed sensor value; and (2) the *number of poisoning samples* in the generated sequence.

Evaluation process. We ran grid tests for both poisoning algorithms for multiple values of: *(i)* target attack magnitude, *(ii)* attack location (explained in the following section), and *(iii)* modeled subsequence length. Each configuration was tested five times, and the metrics were averaged.

4.2 Datasets

Synthetic dataset. For the synthetic data experiments, we used a number of linear dependent sines to model a simple case of correlated system characteristics. The signal amplitude was between −0.5 and 0.5, and distorting Gaussian noise with a mean of zero and a standard deviation of 0.025 was applied to the signal. To simulate the attacks, we increased the signal amplitude by a specified value (attack magnitude). Two different attack locations were tested, the highest point of the signal (SIN_TOP) and the lowest point of the signal (SIN_BOTTOM), as illustrated in Figure 5. The rational behind testing these attack locations was to model two types of malicious signal manipulation. For the SIN_BOTTOM location, the spoofed signal stays in the range of normal signal values, while for SIN_TOP it goes beyond this range. In order to simulate an attacker controlling some of the sensors, the attacks were applied to just one signal.

ICS testbed dataset. For the real-world data experiments, we utilized the popular SWaT dataset [7]. The dataset was collected from the secure water treatment (SWaT) testbed at Singapore University of Technology and Design and has been used in many studies since it was created. The testbed is a scaled-down water treatment plant running a six-stage water purification process. Each process stage is controlled by a PLC with sensors and actuators connected to it. The sensors include flow meters, water level meters, and conductivity analyzers, while the actuators are water pumps, chemical dosing pumps, and inflow valves. The dataset contains 51 attributes capturing the states of the sensors and actuators each second for seven days of recording under normal conditions and four days of recording when the system was under attack (the data for this time period contains 36 attacks). Each attack targets a concrete physical effect, such as overflowing a water tank by falsely reporting a low water level, thus causing the inflow to continue.

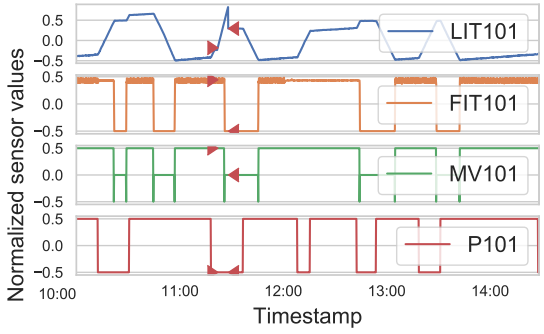

Figure 6: SWaT attack #3. The attacker gradually increases the value of the LIT101 water level sensor in order to cause the tank underflow (red arrows denote the attack's beginning and end).

Following our threat model (see Section 2), we selected seven attacks (3, 7, 16, 31, 32, 33, 36) that involve sensor value manipulations; the attacks are described in Table 1. In these attacks, the

attacker manipulated the water tank level sensor value and reported it to be either below or above the actual level thus causing the PLC to overflow or underflow the tank. The selected attacks represent a number of possible locations and magnitudes. In addition, in attacks #3 and 16 the attacker changes the attacked sensor's value gradually (see Figure 6), while in others the value changes abruptly. Thus, the selected attacks represent a variety of attack methods.

Table 1: SWaT attacks selected for poisoning.

#	Attacked sensor (magnitude)	Modeled sensors	Description	Expected impact
3	LIT-101 (0.83)	LIT101, FIT101, MV101, P101	Increase water level by 1mm every second	Tank underflow; damage P-101
7	LIT-301 (1.0)	LIT301, FIT201, P302	Increase water level above HH	Stop inflow; tank underflow; damage P-301
16	LIT-301 (-1.0)	LIT301, FIT201, P302	Decrease water level by 1mm each second	Tank overflow
31	LIT-401 (-1.0)	LIT401, P302, LIT301, P402	Set LIT-401 to less than L	Tank overflow
32	LIT-301 (1.0)	LIT301, FIT201, P302	Set LIT-301 to above HH	Tank underflow; damage P-302
33	LIT-101 (-1.0)	LIT101, FIT101, MV101, P101	Set LIT-101 to above H	Tank underflow; damage P-101
36	LIT-101 (-1.0)	LIT101, FIT101, MV101, P101	Set LIT-101 to less than LL	Tank overflow

Each attack starts with a water level between L and H
Legend: L - low setpoint value, H - high setpoint value, LL - dangerously low level, HH - dangerously high level.

In the threat model considered, the attacker has access to the real sensory data. However, as we had no access to the testbed, we did not know the real values of the features during the attacks, only the spoofed ones. After trying to reconstruct the real values using a number of heuristic methods, we concluded that the heuristics provided imprecise results and would distort the experiment. Therefore, instead of using the attacks' sensory data from the SWaT test dataset, we simulated the selected attacks by applying the same transformations to the corresponding parts of the training dataset. The data was normalized to the −0.5 - 0.5 range.

4.3 Synthetic Signal Poisoning

In order to be consistent with the ratio of the attack duration to the attacked signal period for attacks in the SWaT dataset, we used sine waves with a period of 500 time steps and attacks with a duration of 40 time steps. In each poisoning attempt the attacker generated two periods of the signal containing the intended poison. The size of the training set was 20 periods (or 10 poisoning samples). All tests used a detection threshold of 0.2 and a stochastic gradient descent optimizer with the learning rate of 0.6. As expected, we observed that with unlimited time, the algorithms could poison the model so that it would accept any target attack (unless it was unachievable due to the tight constraints of the last layer activation). However, unlimited time attacks are usually impractical, as the normal system drift and other environmental and process changes will render the poison ineffective. Also, prolonged attacks are more likely to be detected by a human operator. In order to simplify the presentation and limit the test run time, we set the maximal number of poisoning algorithm iterations to 300. We present the main highlights of the experiments in Table 2 and the algorithms' execution time in Table 3.

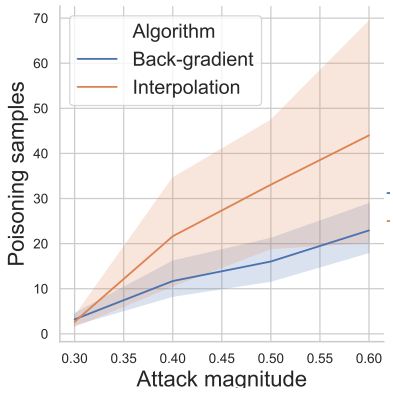

Figure 7: The average number of poisoning samples required to achieve the required attack magnitude for the SIN_BOTTOM location. The shaded areas show the confidence interval for each sequence length.

Poisoning effectiveness. In general, both algorithms successfully generated poisoning for the target attacks. An analysis of the results reveals a number of trends and factors influencing the success of poisoning:

- **The attack magnitude**. As can be seen in Figure 7, there is a dependency between the targeted attack magnitude and the number of poisoning samples required. Greater magnitudes required more points. For many tests, the required amount of poisoning samples was several times larger than the size of the original training set. We observed that for the greatest attack magnitudes there was a slight decline in the number of points required. A possible explanation for that is the attack signal that started from the lowest point of the sine nearly reached the highest point of the sine thus becoming more similar to the original signal.
- **Algorithm**. The back-gradient algorithm produced superior results, obtaining greater magnitudes with less poisoning samples for all tests except for those performed with the sequence length of two (see Figure 7 and Table 2).
- **Sequence length**. In the majority of the tests longer sequence lengths required the attacker to use more poisoning samples.
- **Attack location**. In all tests the attacker was able to produce attacks with greater magnitudes for the BOTTOM location, as evident from Table 2. It should be noted that the maximal possible magnitude for the TOP location was limited to 0.5 due to the constraints of *tanh* activation in the last layer.

4.4 Poisoning Attacks on SWaT

For the simulated SWat attacks, we modeled a small number of the features affected by each attack (see Table 1) with a constrained attacker only able to manipulate a single sensor's data.
Poisoning effectiveness. The results show that a number of factors influenced the poisoning's success:

Table 2: Comparison of maximal attack magnitude achieved by poisoning for different algorithms and locations.

Modeled sequence length	Algorithm/Location			
	Back-gradient		Interpolation	
	BOTTOM	TOP	BOTTOM	TOP
2	0.60(52)	0.30(6)	0.80(6)	0.50(10)
12	0.90(26)	0.40(14)	0.50(106)	0.30(5)
22	0.90(12)	0.40(123)	0.60(98)	0.30(2)
32	0.90(68)	0.40(67)	0.40(89)	0.30(106)
42	0.90(85)	0.35(45)	-	-

[1] The number of poisoning samples for the attack is shown in parentheses.
[2] The maximal possible magnitude for the TOP location was 0.5.

Table 3: Iteration execution time (in seconds).

Algorithm	Sequence length				
	2	12	22	32	42
Interpolative	27.8	41.0	46.6	40.2	45.6
Back-gradient	52.8	70.1	75.3	66.4	83.9

The tests were run on AWS c5n.large instance.

Table 4: SWaT attacks' poisoning results for the interpolation algorithm.

Seq. length	Attack #						
	3[1]	7	16[1]	31[1]	32	33	36
2	✓(1)	✓(1)	✓(3)	✓(11)	✓(1)	✓(2)	✓(1)
10	✓(2)	✓(1)	✗	✓(31)	✓(3)	✓(3)	✓(2)
20	✓(4)	✓(6)	✗	✓(29)	✓(5)	✓(11)	✓(35)
30	✓(8)	✓(15)	✗	✓(39)	✓(5)	✓(19)	✓(32)

✓ denotes successful poisoning with the number of poisoning samples for the attack shown in parentheses; ✗ denotes failure to generate the poisoning within 300 iterations.
[1] The tests were performed with a threshold of 0.1.

Table 5: SWaT attacks' poisoning results for the back-gradient algorithm.

Seq. length	Attack #						
	3[1]	7	16[1]	31[1]	32	33	36
2	✓(1)	✓(1)	✓(1)	✗	✓(1)	✓(1)	✓(1)
10	✓(1)	✓(9)	✗	✗	✓(2)	✓(6)	✓(4)
20	✓(11)	✓(21)	✗	✗	✓(5)	✓(9)	✓(8)
30	✓(9)	✓(11)	✗	✗	✓(10)	✗	✓(9)

[1] The tests were performed with a threshold of 0.1

- **The attack magnitude, location, and abruptness**. Attacks #3 and 16, which are characterized by a gradual change of the spoofed signal, were poisoned using only one or two samples for all sequence lengths when using the default threshold of 0.2. The main reason for the ease in poisoning was the low initial residual of the attack stemming from its subtle character. Therefore, we tested poisoning for both attacks with the threshold of 0.1 (see below).
- **Sequence length.** The sequence length influences the model's resilience. Increasing it caused the attacker to use more poisoning samples. Figure 8 illustrates the change in the detection residual of attacks #16 and 31 with the introduction of additional poisoning samples.
- **Detection threshold.** The detection threshold has a strong influence on the model's resilience as well. Decreasing it to 0.1 (tested with attacks #3, 16, and 31) increased the number of poisoning samples required and caused many of these attacks to fail. We examined the impact of the detection threshold value on the number of false positives for the entire SWaT dataset. We found that when using the detection methodology described in [14] and [15] the threshold could be as low as 0.05 without introducing false positives.
- **Algorithm**. For the SWaT tests, the back-gradient algorithm did not perform better than the interpolation algorithm and required more poisoning samples in some cases. Our analysis suggests that the difference between the synthetic and SWaT dataset results from the difference in the poison generation strategy of the two algorithms. The interpolation algorithm uses the largest step possible in the direction of the attack, while the back-optimization takes smaller steps in the direction of the calculated gradients. The results show that each strategy has its place. In the case of closely related (linearly dependent) synthetic signals, smaller steps are preferred. In the case of more loosely related SWaT sensors (see Figure 6), larger steps provide better results. An optimal learning rate scheduling algorithm combining the strength of both approaches will be a topic of future research.

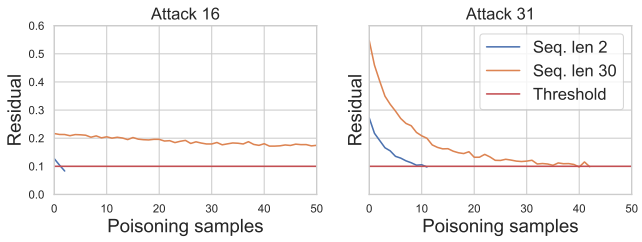

Figure 8: Attack detection residual change with poisoning samples addition (presented for attacks #16 and 31), and a threshold of 0.1 (the red line). With a sequence length of two the poisoning succeeds easily, whereas the sequence length of 30 provides strong poisoning resistance and causes the poisoning to fail for attack #16.

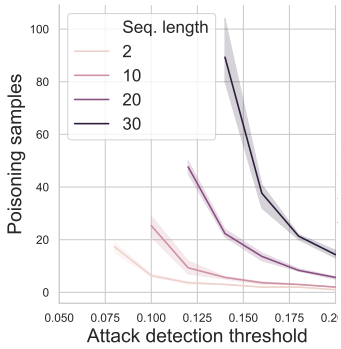

Figure 9: The average number of poisoning samples required to achieve the SWaT attack #7 under the given detection threshold. Lower thresholds require more points. It was not possible to achieve the attack for thresholds lower than the left-most point of each graph.

4.5 Poisoning Attack Mitigation

Our findings suggest that **decreasing the detection threshold** is an effective means of poisoning mitigation. Figure 9 illustrates this effect for the attack #7, the same behaviour was observed with other

attacks. At the same time, decreasing it too much will introduce false positives by the attack detection model.

Increasing the sequence length seems to be another effective poisoning countermeasure (see Tables 4 and 5). To evaluate the influence of this length increase on the overall anomaly detection performance, we performed the detection of all attacks (not only the selected ones) in the SWaT dataset using different sequence lengths and compared the average $F1$ detection score from five runs of each configuration. As can be seen in Table 6, increasing the length results in just a 2-4% decrease in the detection score.

As a third form of mitigation, we propose using **two detector models** with different sequence lengths - one short and one long. The models process the input in parallel and either one can detect an anomaly. In this setup, the attacker will have to produce poisoning that influences both models simultaneously. The differences in modeling that result from different sequence lengths should prove challenging to the attacker. We tested this hypothesis on the synthetic data with two models, one using the sequence length of two and another with a length of 22, using the interpolation algorithm. As shown in Table 7, the dual model detector exhibited stronger resilience than each component on its own. This promising direction will be studied in more depth in future research.

Table 6: Average F1 for attack detection on the SWaT dataset for different modeled sequence lengths.

Modeled sequence length	1	10	20	30	40
Average detection $F1$	0.846	0.826	0.831	0.828	0.824

Table 7: Dual detector poisoning results.

Seq. length	Second seq. length	Attack magnitude	
		0.4	0.5
2	-	✓(5)	✓(8)
-	22	✓(3)	✓(5)
2	22	✓(12)	✓(62)

5 RELATED WORK

A number of recent studies have focused on evasion attacks on CPS anomaly detectors. In [3], the authors showed that generative adversarial networks (GANs) can be used for real-time learning of an unknown ICS anomaly detector (more specifically, a classifier) and for the generation of malicious sensor measurements that will go undetected. The research in [4] presents an iterative algorithm for generating stealthy attacks on linear regression and feedforward neural network-based detectors. The algorithm uses mixed-integer linear programming (MILP) to solve this problem. For NN detectors, the algorithm first linearizes the network at each operating point and then solves the MILP problem. The paper demonstrates a successful evasion attack on a simulated Tennessee Eastman process.

Recently, Erba *et al.* [2] demonstrated a successful real-time evasion attack on an autoencoder-based detection mechanism in water distribution systems. The authors of [2] considered a white-box attacker that generates two different sets of spoofed sensor values: one is sent to the PLC, and the other is sent to the detector.

The most recent paper in this area [33] also focused on an adversary that can manipulate sensor readings sent to the detector. The authors showed that such attackers can conceal most of the attacks present in the SWaT dataset. Our study differs from these studies in a number of ways. First and foremost, all of the above-mentioned papers examined evasion attacks, while our research focuses on poisoning attacks. Second, [2, 3, 33] considered a threat model in which the attacker manipulates the detector's input data *in addition* to manipulating the sensor data fed to the PLC. Such a model provides a lot of freedom for the adversary to make changes to both types of data. Our threat model considers a significantly more constrained attacker that can only change the sensory data that is provided **both** to the PLC and the detector.

A few recently published papers have studied poisoning attacks, however the authors considered them in a different context. The study performed by Muñoz-González *et al.* [20] was the first one to successfully demonstrate poisoning attacks on multiclass classification problems. It also was the first to suggest generating poisoning data using back-gradient optimization. Our research extends this method to semi-supervised multivariate time series regression tasks in the online training setting and evaluates the robustness of an autoencoder-based detector to such attacks.

Shafani *et al.* [26] and Suciu *et al.* [28] studied clean-label poisoning of classifiers. In targeted clean-label poisoning, the attacker does not have control of the labels for the training data and changes the classifier's behavior for a specific test instance without degrading its overall performance for other test inputs. These studies differ significantly from ours, both in terms of the learning task to be poisoned (classification vs. regression) and the domain (images vs. long interdependent multivariate time sequences).

Madani *et al.* [18] studied adversarial label contamination of autoencoder-based intrusion detection for network traffic monitoring. Their research considered a black-box attacker that gradually adds *existing* malicious samples to the training set, labeling them as normal. Such a setting is very different from the one studied in our work. First, we consider semi-supervised training; thus, there is no labeling involved. Second, we explore algorithms for *generating* adversarial poisoning samples that will direct the detector's outcome towards the target goal.

To summarize, although some of the previous research has dealt with related topics or domains, to the best of our knowledge, this study is the first one to address poisoning attacks on multivariate regression learning algorithms and specifically on online-trained physics-based anomaly and intrusion detectors in CPSs.

6 CONCLUSIONS

The reliability of NN-based cyber attack detectors depends on their resilience to adversarial data attacks. In this study, we presented two algorithms for poisoning such detectors under a threat model relevant to online-trained ICSs. The algorithms were evaluated on two datasets and found capable of poisoning the detectors both using artificial and real testbed data. To the best of our knowledge, this is the first time adversarial poisoning of multivariate NN regression-based tasks with constraints is presented.

Our results also point out a number of factors influencing the robustness of the examined detectors to poisoning attacks. First, in most of attacks the attacker had to generate long sequences of poisoning samples, often exceeding the amount of training data. This creates practical difficulty in carrying out such attacks without

being detected or affected by natural process changes. Therefore, we can point out at a certain inherent robustness of the detectors. In addition, the detectors can leverage the constraints of the NN architecture used, to prevent the attacker from drifting the model to accept arbitrary values even with the unlimited attack time. In our experiments, we used the *tanh* activation in the last layer for such protection.

In addition, we evaluated three poisoning mitigation techniques: (1) decreasing the detection threshold, (2) increasing the sequence length, and (3) using dual detectors. The mitigations were found effective and did not degrade the detection metrics significantly.

Some limitations and future directions of this study are worth noting. First, this study only evaluated autoencoder-based detectors; studying the robustness of other NN architectures is a topic for future research.

This study could also be extended to evaluate the algorithms on more complex data involving attackers controlling multiple sensors. Another important issue for future studies is increasing the efficiency of the poisoning algorithms proposed. Currently, they require significant computational time, thus making them inappropriate for real-time poisoning. Lastly, in this study, we used a data-only approach for the algorithms' verification. Validating the findings in a real-world testbed would be a natural next stage of this research, possibly after finding a way to accelerate poisoning sequence generation.

Acknowledgments

This research was partially supported by the CONCORDIA project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 830927; by the PRIN 2017 project RexLearn (grant no. 2017TWNMH2), funded by the Italian Ministry of Education, University and Research; and by BMK, BMDW, and the Province of Upper Austria in the frame of the COMET Programme managed by FFG in the COMET Module S3AI.

REFERENCES

[1] Chuadhry Mujeeb Ahmed, Jianying Zhou, and Aditya P Mathur. 2018. Noise matters: Using sensor and process noise fingerprint to detect stealthy cyber attacks and authenticate sensors in cps. In *Proc. 34th Annual Computer Security Applications Conference*. 566–581.

[2] Alessandro Erba, Riccardo Taormina, Stefano Galelli, Marcello Pogliani, Michele Carminati, Stefano Zanero, and Nils Ole Tippenhauer. 2019. Real-time evasion attacks with physical constraints on deep learning-based anomaly detectors in industrial control systems. *arXiv preprint arXiv:1907.07487* (2019).

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

[4] Amin Ghafouri, Yevgeniy Vorobeychik, and Xenofon Koutsoukos. 2018. Adversarial regression for detecting attacks in cyber-physical systems. In *Proc. 27th Int'l Joint Conf. Artificial Intelligence*. 3769–3775.

[5] Jairo Giraldo, Esha Sarkar, Alvaro A Cardenas, Michail Maniatakos, and Murat Kantarcioglu. 2017. Security and privacy in cyber-physical systems: A survey of surveys. *IEEE Design & Test* 34, 4 (2017), 7–17.

[6] Jairo Giraldo, David Urbina, Alvaro Cardenas, Junia Valente, Mustafa Faisal, Justin Ruths, Nils Ole Tippenhauer, Henrik Sandberg, and Richard Candell. 2018. A survey of physics-based attack detection in cyber-physical systems. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 76.

[7] 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 *Int'l Conference on Critical Information Infrastructures Security*. Springer, 88–99.

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

[9] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[10] Amir Herzberg and Yehonatan Kfir. 2019. The chatty-sensor: a provably-covert channel in cyber physical systems. In *Proc. 35th Annual Computer Security Applications Conference*. 638–649.

[11] Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. 2017. Cyber-physical systems security âĂŤ A survey. *IEEE Internet of Things Journal* 4, 6 (2017), 1802–1831.

[12] Jun Inoue, Yoriyuki Yamagata, Yuqi Chen, Christopher M Poskitt, and Jun Sun. 2017. Anomaly detection for a water treatment system using unsupervised machine learning. In *Int'l Conf. Data Mining Workshops*. IEEE, 1058–1065.

[13] Jonguk Kim, Jeong-Han Yun, and Hyoung Chun Kim. 2019. Anomaly detection for industrial control systems using sequence-to-sequence neural networks. In *Computer Security*. Springer, 3–18.

[14] Moshe Kravchik and Asaf Shabtai. 2018. Detecting cyber attacks in industrial control systems using convolutional neural networks. In *Proc. 2018 Workshop on Cyber-Physical Systems Security and PrivaCy*. ACM, 72–83.

[15] Moshe Kravchik and Asaf Shabtai. 2019. Efficient cyber attacks detection in industrial control systems using lightweight neural networks. *arXiv preprint arXiv:1907.01216* (2019).

[16] Qin Lin, Sridha Adepu, Sicco Verwer, and Aditya Mathur. 2018. TABOR: a graphical model-based approach for anomaly detection in industrial control systems. In *Proc. 2018 on Asia Conf. Comp. Comm. Sec.* ACM, 525–536.

[17] Dougal Maclaurin, David Duvenaud, and Ryan Adams. 2015. Gradient-based hyperparameter optimization through reversible learning. In *Int'l Conference on Machine Learning*. 2113–2122.

[18] Pooria Madani and Natalija Vlajic. 2018. Robustness of deep autoencoder in intrusion detection under adversarial contamination. In *Proc. 5th Annual Symp. and Bootcamp on Hot Topics in the Science of Security*. ACM, 1.

[19] Robert Mitchell and Ing-Ray Chen. 2014. A survey of intrusion detection techniques for cyber-physical systems. *Comput. Surveys* 46, 4 (2014), 55.

[20] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. 2017. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proc. 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 27–38.

[21] Mykola Pechenizkiy, Jorn Bakker, I Žliobaitė, Andriy Ivannikov, and Tommi Kärkkäinen. 2010. Online mass flow prediction in CFB boilers with explicit detection of sudden concept drift. *ACM SIGKDD Explorations Newsletter* 11, 2 (2010), 109–116.

[22] Waldir Ribeiro Pires, Thiago H de Paula Figueiredo, Hao Chi Wong, and Antonio Alfredo Ferreira Loureiro. 2004. Malicious node detection in wireless sensor networks. In *18th Int'l Parallel and Distributed Processing Symp.* IEEE, 24.

[23] Gelli Ravikumar, Burhan Hyder, and Manimaran Govindarasu. 2020. Next-Generation CPS Testbed-based Grid Exercise-Synthetic Grid, Attack, and Defense Modeling. In *2020 Resilience Week (RWS)*. IEEE, 92–98.

[24] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. 2018. Generic black-box end-to-end attack against state of the art API call based malware classifiers. In *Int'l Symp. Res. Attacks, Intrusions, and Defenses*. Springer, 490–510.

[25] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J Doug Tygar. 2009. Antidote: understanding and defending against poisoning of anomaly detectors. In *Proc. 9th ACM SIGCOMM conference on Internet measurement*. 1–14.

[26] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Adv. Neural Inf. Proc. Sys.* 6103–6113.

[27] Elaine Shi and Adrian Perrig. 2004. Designing secure sensor networks. *IEEE Wireless Communications* 11, 6 (2004), 38–43.

[28] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. 2018. When does machine learning FAIL? generalized transferability for evasion and poisoning attacks. In *27th USENIX Sec. Symp.* 1299–1316.

[29] Riccardo Taormina and Stefano Galelli. 2018. Deep-learning approach to the detection and localization of cyber-physical attacks on water distribution systems. *Journal of Water Resources Planning and Management* 144, 10 (2018), 04018065.

[30] Riccardo Taormina, Stefano Galelli, Nils Ole Tippenhauer, Elad Salomons, Avi Ostfeld, Demetrios G Eliades, Mohsen Aghashahi, Raanju Sundararajan, Mohsen Pourahmadi, M Katherine Banks, et al. 2018. Battle of the attack detection algorithms: Disclosing cyber attacks on water distribution networks. *Journal of Water Resources Planning and Management* 144, 8 (2018), 04018048.

[31] Xiaojun Zhou, Zhen Xu, Liming Wang, Kai Chen, Cong Chen, and Wei Zhang. 2018. APT attack analysis in SCADA systems. In *MATEC Web of Conferences*, Vol. 173. EDP Sciences, 01010.

[32] Giulio Zizzo, Chris Hankin, Sergio Maffeis, and Kevin Jones. 2019. Adversarial machine learning beyond the image domain. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–4.

[33] Giulio Zizzo, Chris Hankin, Sergio Maffeis, and Kevin Jones. 2019. Intrusion Detection for Industrial Control Systems: Evaluation Analysis and Adversarial Attacks. *arXiv preprint arXiv:1911.04278* (2019).