# Practical Evaluation of Poisoning Attacks on Online Anomaly Detectors in Industrial Control Systems*

Moshe Kravchik[a,b,*], Luca Demetrio[c], Battista Biggio[c] and Asaf Shabtai[a]

[a]Ben-Gurion University of the Negev
[b]Rafael Advanced Defense Systems LTD
[c]University of Cagliari

ARTICLE INFO

*Keywords*:
Anomaly detection
industrial control systems
autoencoders
adversarial machine learning
poisoning attacks
adversarial robustness

ABSTRACT

Recently, neural networks (NNs) 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 evolution of the monitored signals over time. 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 to allow cyber attacks to stay undetected at test time. Previous work explored the ability to generate adversarial samples that fool anomaly detection models in ICSs but without compromising their training process. With this research, we are the first to demonstrate such poisoning attacks on ICS cyber attack online detectors based on neural networks. We propose two distinct attack algorithms, namely, interpolation- and back-gradient-based poisoning, and demonstrate their effectiveness. The evaluation is conducted on diverse data sources: synthetic data, real-world ICS testbed data, and a simulation of the Tennessee Eastman process. This first practical evaluation of poisoning attacks using a simulation tool highlights the challenges of poisoning dynamically controlled systems. The generality of the proposed methods under different NN parameters and architectures is studied. Lastly, we propose and analyze some potential mitigation strategies.

## 1. Introduction

Neural networks (NN) exhibit impressive performance and are now being assisted in many different areas such as medical diagnosis, computer vision, autonomous driving, and cyber attack detection. In particular, these technologies have been introduced to monitor and detect possible incoming cyber attacks that target Industrial Control Systems (ICS), a subset of cyber-physical systems (CPS) [11, 12, 14, 28, 29, 34, 53, 52, 55, 56, 18, 23, 25, 40]. Defending these systems is extremely important, since ICSs are central to many areas of industry, energy production, and critical infrastructure, and they are exposed to external threats as they need to be remotely accessible by operators.

However, with the advent of *adversarial machine learning* [5], researchers have shown how skilled attackers can hinder the performances of deep neural networks, by feeding them with the so-called *adversarial examples*, making them unable to fulfill their intended task. These attacks are already targeting many fields of application, from the image classification [4, 51, 50, 48], to malware detection [50, 46, 9, 8], and to network intrusion detection [47, 37]. NN-based algorithms that detect cyber-threats of ICSs are vulnerable to adversarial attacks as well. When an attacker has access to the target only at test time, they can craft *evasion attacks*, by manipulating input samples that are misclassified by the target model [11, 56, 29]. In the context of threat detectors, this is equivalent to bypass detection, with the subsequent success of the intended attack against the victim. However, it is difficult to cause a significant physical impact on the controlled process while evading a NN detector [29]. Hence, to maximize the damage, the attacker might attempt *poisoning attacks*, where they introduce adversarial data when the model is being trained. Such injected data influences the model by either degrading its performances and making it unusable, or by allowing future evasion attacks. The importance of poisoning attack research increases in light of the popularity of monitoring system operating in an 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 [56] 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 the detector's retraining with the new data. Another ICS online trained predictor is described in [42], where it is used to predict mass flow in an industrial boiler and is retrained to address the concept drift caused by a nonstandardized feeding process. Such online retraining provides the adversary with the opportunity to poison the model; however, the state of the art lacks the study of the efficacy of poisoning techniques in this context.

**Problem statement.** The threat model assumed in our research considers an adversary whose goal is to change a physical process of the targeted ICS, which includes an online-trained anomaly detector. We model the presence of an adversary that has gained control of a sensor or a number (subset) of sensors, and can falsify the sensors' readings. Spoofed

*A preliminary version of this paper was presented at the ACM SAC 2021 conference.

*Corresponding author
moshekr@post.bgu.ac.il (M. Kravchik); luca.demetrio93@unica.it (L. Demetrio); battista.biggio@unica.it (B. Biggio); shabtaia@bgu.ac.il (A. Shabtai)
ORCID(s): 0000-0001-8171-3755 (M. Kravchik); 0000-0001-7752-509X (L. Demetrio); 0000-0001-7752-509X (B. Biggio); 0000-0003-0630-4059 (A. Shabtai)

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. The dynamic control of the ICS presents an additional challenge to the attacker. The process reacts to the introduced poison in ways that might increase the poison detectability and decrease the attack effectiveness. While the majority of prior studies validate the detector and attacks using static datasets, we hypothesize that the dynamic response has a major impact on the ability to execute poisoning attacks. To the best of our knowledge, no such dynamic system poisoning study was conducted before. Our study aims to answer the following research questions: (1) What algorithms can be used to effectively generate poisoning input for an NN-based ICS anomaly detector operating in online training mode? (2) How robust are the state-of-the-art detectors [52, 11, 29] to such poisoning attacks? (3) How generic and transferable the attacks generated by the proposed algorithms are? (4) How effective the proposed attacks are against a dynamic ICS actively countering them? and (5) 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 conduct an evaluation of the attacks' transferability to different NN parameters and architectures. *vi)* We implement the interpolation poisoning attack in a test environment based on the simulated Tennessee Eastman process and study the effectiveness and limitations of the proposed algorithms with regard to dynamic systems. To the best of our knowledge, this is the first study on dynamic system poisoning under control and detector constraints. *vii)* We propose several mitigation techniques against poisoning attacks. *viii)* The implementation of both algorithms and the evaluation test code are open source and freely available.[1]

This study is an extension of our previous work [27]. While the preliminary version focused on the poisoning algorithms and their evaluation on synthetic and static data, this paper (1) extends the evaluation of the proposed poisoning attack to two additional real-world benchmark datasets, (2) presents a methodology for practical evaluation of the proposed algorithms, (3) proposes a novel five-step method for the poisoning algorithms application to dynamic systems including the use of two additional NN predictors, its implementation and evaluation, and finally (4) conducts an extensive transferability study.

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

## 2. Industrial Control Systems

An Industrial Control System (ICS) is a 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. ICSs are typically attacked either at the sensor or at the PLC level in the remote segment [15, 16].

*Intrusion detection systems* The 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 [16, 38, 22, 15]. 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. This approach has recently become very popular due to the ability of NNs to model complex multivariate systems with nonlinear dependencies among the variables [28, 52, 34, 53, 11, 29]. Various NN architectures have been used to this end, including convolutional NNs, recurrent NNs, feedforward NNs, and autoencoders [19]. While our attacks *are general enough to be applied against any kind of NN architectures*, in this paper we opted to evaluate them against autoencoders due to their increasing popularity [52, 11, 29].

*Threat Model* To evaluate the robustness of such detectors to poisoning attacks, we consider a malicious sensor threat model widely studied in the wireless sensor network domain [43, 49], in a related ICS research [20], and used in a real-world attack-defense exercise for smart grids [45]. 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. We 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 modeling an attacker that controls both the sensors and the internal network of the remote segment or even the network of the control segment, as considered in previous studies such as in [11, 56]. Moreover, the setting we considered presents more constraints and challenges to the attacker, who must achieve their goals with a single location of data manipulation. Finally, as this is the first work to demonstrate that is possible to perform poisoning attacks against cyber attack detectors for ICSs, we assume a white-box attack scenario in which the attacker has full knowledge of the detector's model (i.e,. learning algorithm and hyperparameters) [6]. Related researches [14, 55, 2] use the white-box threat model as well. Since ICSs are commonly attacked by state-sponsored actors, assuming an attacker that has knowledge about the attacked model is not unrealistic. 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

In this section we describe 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. In 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 their poisoning samples to be used for retraining. If the detector's training schedule is unknown to the attacker, the attacker 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 permitted. This makes our poisoning attack more challenging, as the poisoning samples themselves will have to stay undetected.

We first discuss the considered anomaly detector and the used notation (Section 3.1), then proceed to describe our poisoning attack strategies (Section 3.2).

### 3.1. Anomaly Detection with Autoencoder

In this research, we focus on reconstruction-based anomaly detectors, utilized in [18, 23, 28, 34, 53, 52, 29], which are most commonly used with ICSs [11]. The general idea behind such detectors is to train the model to reconstruct a sample from its reduced latent representation, using normal data. At test time, normal samples should be reconstructed accurately, while anomalous data should have larger reconstruction errors. In ICSs, the detectors usually operate on sensor readings and actuator states. There are two stages to detection: (1) the model produces its prediction for the observed input values and the reconstruction error is calculated, and (2) the error is compared to a threshold. To avoid false alarms, it is common to consider a detection time window (e.g., the last ten samples) and raise an alert only if the error is above the threshold for all samples in that window [52, 11, 29].

We denote with $\mathbf{x}_t \in \mathbb{R}^d$ the $d$-dimensional vector consisting of the sensor measurements and actuator states observed by the PLC at time $t$, and with $\mathbf{X}_{t,S} = (\mathbf{x}_{t-S}, \dots, \mathbf{x}_t) \in \mathbb{R}^{d \times S}$ - a sequence of such signals from time $t - S$ to time $t$, being $S$ the *sequence length*. We consider a cyber attack detector $f$ based on an NN model (e.g., autoencoder) that reconstructs such values at each time point $t$ as:

$$\hat{\mathbf{X}}_{t,S} = f_{\mathbf{w}}(\mathbf{X}_{t,S}) \tag{1}$$

where $\mathbf{w}$ are the model parameters (i.e., weights) and $\hat{\mathbf{X}}_{t,S}$ is the reconstructed vector. The model parameters $\mathbf{w}$ are learned at training time by minimizing a loss function $\mathcal{L}(\mathcal{D}_{\text{tr}}, \mathbf{w})$ on the training data $\mathcal{D}_{\text{tr}} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$, with $N >> S$. In this work, we use the mean squared error (MSE) as the loss function, considering each predicted sequence separately, and it is computed as:

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

At detection time, a test input $\mathbf{X}_{t,K} = (\mathbf{x}_{t-K}, \dots, \mathbf{x}_t)$, i.e., a sequence of $K$ vectors, where $K$ is defined as the detection window size at time $t$ (possibly with $K < S$ for prompt detection) is compared against the corresponding predictions to compute the residuals. The residuals are measured as the Chebyshev distance - the largest deviation between the real and predicted values across all features (sensors and actuators), and it is defined as follows:

$$g(\mathbf{X}_{t,K}) = \min_{k \in \{t-K,\dots,t\}} \|\hat{\mathbf{x}}_k - \mathbf{x}_k\|_\infty \tag{3}$$

To avoid false positives, the deviation between the predicted and observed signals should be significant enough for the entire length of $K$. Therefore, an attack is detected if the

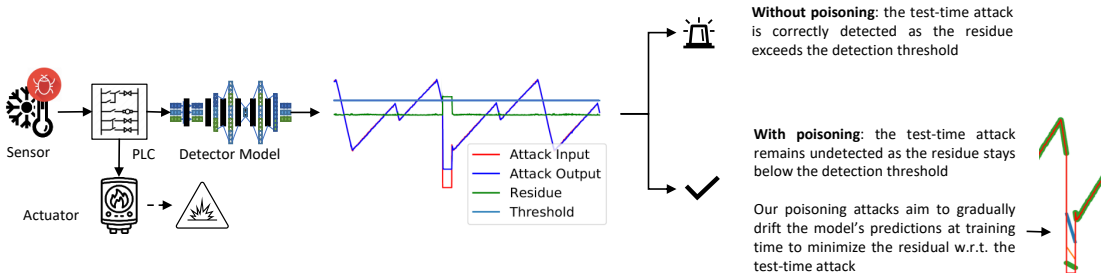


**Figure 1:** ICS cyber attack detector under a poisoning attack.

smallest residual in this sequence denoted as $g$ exceeds the detection threshold $\tau$, and we denote such detection function as:

$$y(\mathbf{X}_{t,K}) = \begin{cases} 'attack' & \text{if } g(\mathbf{X}_{t,K}) > \tau, \\ 'normal' & \text{otherwise.} \end{cases} \tag{4}$$

### 3.2. Poisoning Attacks

The poisoning attack considered in this work aims to allow a specific attack $\mathbf{X}^a = (\mathbf{x}^a_1, \ldots, \mathbf{x}^a_Q)$ of length $Q$ to stay undetected at test time, by injecting a carefully-optimized *sequence* of $M$ *poisoning samples* $\mathcal{D}_p = (\mathbf{X}^p_1, \ldots, \mathbf{X}^p_M)$ into the training data used by the detector. Note that here each poisoning sample $\mathbf{X}^p_k = (\mathbf{x}^p_{k1}, \ldots, \mathbf{x}^p_{kT})$ contains $T$ poisoning points, aiming to capture the natural periodicity of the given physical signals in order to to stay undetected. Our poisoning attack is successful if the detector, trained on $\mathcal{D}_{\text{tr}} \cup \mathcal{D}_p$, does not detect $\mathbf{X}^a$ at test time, while also not raising false alarms on normal data. The proposed poisoning scenario is summarized in Figure 1. Our attack can be formulated as a bi-level optimization problem:

$$\mathcal{D}^\star_p \in \arg\min_{\mathcal{D}_p} L(\mathbf{X}^a, \mathbf{w}^\star) \tag{5}$$

$$\text{s.t.} \quad \mathbf{w}^\star \in \arg\min_{\mathbf{w}} \mathcal{L}(\mathcal{D}_{\text{tr}} \cup \mathcal{D}_p, \mathbf{w}) \tag{6}$$

$$g(\mathcal{D}_p) \preceq \tau \tag{7}$$

$$g(\mathcal{D}_{\text{val}}) \preceq \tau \tag{8}$$

where the outer problem in Eq. 5 corresponds to having the attack sample undetected, the inner problem in Eq. 6 amounts to training the autoencoder on the poisoned training data, and the constraints in Eqs. 7-8 respectively require the poisoning samples and normal data (drawn from a validation set) to always stay below the detection threshold $\tau$. Note that the poisoning samples only influence the outer objective indirectly, through the choice of the optimal parameters $\mathbf{w}^\star$ learned from the poisoned training data. The outer objective (the loss on the attack input) $L(\mathbf{X}^a, \mathbf{w})$ is computed as done for the training loss in Eq. 2, i.e., by summing the mean squared errors computed on sequences of the modeled length $S$:

$$L(\mathbf{X}^a, \mathbf{w}) = \sum_{t=S+1}^{Q} \sum_{i=t-S}^{t} \|\hat{\mathbf{x}}^a_i - \mathbf{X}^a_i\|_2^2. \tag{9}$$

Note that while $\mathcal{L}$ denotes the learner's objective function (possibly including regularization), we use $L(D, \mathbf{w})$ to denote only the loss incurred when evaluating the model with weights $\mathbf{w}$ on the samples in D. 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 $\mathbf{X}^p_k$ at a time, and add it to the poisoning set $\mathcal{D}_p$ iteratively.

#### 3.2.1. *Back-gradient optimization attack*

One approach for solving Problem 5-8 is to use gradient descent. To keep notation clean, we denote below the poisoning sample $\mathbf{X}^p_k$ with $x_c$, and the outer objective with $\mathcal{A}(\mathbf{w}^\star(x_c))$, to clarify that it only depends implicitly on $x_c$ 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:

$$\nabla_{x_c} \mathcal{A} = \frac{\delta \mathbf{w}^\star}{\delta x_c}^T \nabla_{\mathbf{w}} L\,, \tag{10}$$

where the term $\frac{\delta \mathbf{w}^\star}{\delta x_c}$ captures the change induced in the optimal parameters $\mathbf{w}^\star$ of the autoencoder due to the injection of the poisoning sample $x_c$ 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 [39]. 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 [36], as suggested in [39]. The core idea of this approach is an iterative backwards calculation of both the weights' updates and $\frac{\delta \mathbf{w}^\star}{\delta x_c}$, 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 [39]).

Algorithm 1 starts with initializing the derivatives of the loss relative to the attack input and the weights of the trained

**Algorithm 1** Find the gradient of the loss on attack input w.r.t. the poisoning sample using back-gradient descent.

**Input:** $\mathbf{w}$, the trained model's weights; $\alpha$, the learning rate; $x_a$, the attack input; $x_c$, the poisoning sample; $L$, the loss function; $\mathcal{L}$, the learner's objective; $T$, the number of iterations.

**Output:** $\nabla_{x_c}\mathcal{A}$, the gradient of the loss.

1: **function** GETPOISONGRAD($\mathbf{w}, \alpha, x_a, x_c, L, \mathcal{L}, T$)
2: $dx_c \leftarrow 0$
3: $d\mathbf{w} \leftarrow \nabla_{\mathbf{w}} L(x_a, \mathbf{w})$
4: **for** $t = T$ to 1 **do**
5: $dx_c \leftarrow dx_c - \alpha d\mathbf{w} \nabla_{x_c} \nabla_{\mathbf{w}} \mathcal{L}(x_c, \mathbf{w}_t)$
6: $d\mathbf{w} \leftarrow d\mathbf{w} - \alpha d\mathbf{w} \nabla_{\mathbf{w}} \nabla_{\mathbf{w}} \mathcal{L}(x_c, \mathbf{w}_t)$
7: $gr \leftarrow \nabla_{\mathbf{w}} \mathcal{L}(x_c, \mathbf{w}_t)$
8: $\mathbf{w}_{t-1} \leftarrow \mathbf{w}_t + \alpha gr$
9: **return** $dx_c$

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 $\nabla_{x_c}\mathcal{A}$ 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 ($\lambda$) dynamic decay used in the algorithm's implementation from the description of Algorithm 2. With the dynamic decay, $\lambda$ 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*.

**Algorithm 2** Compute the poisoning sequence set $\mathcal{D}_p$ with the Back-gradient Optimization Attack.

**Input:** $\mathcal{D}_{tr}$, the training dataset; $\mathcal{D}_{val}$, the validation dataset; $\alpha$, learning rate; $\lambda$, the adversarial learning rate; $x_a$, the attack input; $X_c^0$, the initial poisoning sample; $L$, the loss function; $\mathcal{L}$, the learner's objective; M, the maximum number of iterations.

**Output:** $\mathcal{D}_p$, poisoning sequence set.

1: $\mathcal{D}_p \leftarrow \{\}$
2: $decay \leftarrow 0.9$
3: $eps \leftarrow 10^{-5}$ ▷ Minimal allowed $\lambda$
4: $orig\lambda \leftarrow \lambda$
5: **for** $i = 1$ to M **do**
6: $(\mathbf{w}, alerts) \leftarrow train\_test(\mathcal{D}_{tr}, \mathcal{D}_{val}, x_a, \mathcal{D}_p, X_c^i)$
7: **if** $alerts == 0$ **then**
8: $\mathcal{D}_p \leftarrow \mathcal{D}_p \cup \{X_c^i\}$ ▷ Add current poison
9: **break**
10: $dx_c \leftarrow getPoisonGrad(\mathbf{w}, \alpha, x_a, X_c^i, L, \mathcal{L})$
11: $X_c^{i+i} \leftarrow X_c^i - \lambda \cdot dx_c / \max(dx_c)$
12: ▷ Check that the new poison does not generate alerts
13: $(\mathbf{w}, alerts) \leftarrow train\_test(\mathcal{D}_{tr}, \mathcal{D}_{val}, x_a, \mathcal{D}_p, X_c^{i+i})$
14: **if** $alerts > 0$ **then**
15: $\mathcal{D}_p \leftarrow \mathcal{D}_p \cup \{X_c^i\}$ ▷ Add previous poison
16: $\lambda \leftarrow decay \cdot \lambda$ ▷ Adjust learning rate
17: $X_c^{i+i} \leftarrow X_c^i$ ▷ Revert to last good poison
18: **if** $\lambda <= eps$ **then**
19: **break** ▷ Can't find anymore poisons
20: **else**
21: $\lambda \leftarrow orig\lambda$
22: **return** $\mathcal{D}_p$

**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 backward, and must be taken into account when creating the poisoning input, as this input must therefore be much longer than the sequence of points changed during the targeted 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.

The $\mathcal{WM}$ illustrated in Figure 2 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 overlapping subsequences and appends the model with operations that combine the results of individual predictions and calculate

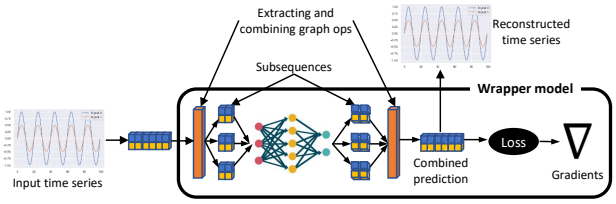


**Figure 2:** Wrapper model that allows for calculating gradients and optimizing the poisoning samples for arbitrary long input sequence based on an original model predicting short sequences.

**Algorithm 3** Compute the poisoning sequence set $\mathcal{D}_p$ with the Interpolation Attack.

**Input:** $\mathcal{D}_{tr}$, the training dataset; $\mathcal{D}_{val}$, the validation dataset; $\delta$, the decay rate; $x_a$, the attack input; $X_c^0$, the initial poisoning sample.

**Output:** $\mathcal{D}_p$, the poisoning sequence set.

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

the combined output. The $\mathcal{WM}$ allows us to calculate the gradients and optimize the adversarial input for an arbitrarily long input sequence.

#### *3.2.2. 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 targeted attack is tested (lines 12-15). Otherwise, the interpolation step is decreased and the interpolation is recalculated (lines 10-11). The algorithm continues until the attack is not detected or the interpolation step becomes too small.

## 4. Evaluation Using Static Datasets

In this section, we present the experimental results of the adversarial attacks evaluation in the static data setup that allows for testing of a wide range of parameters in a stable and repeatable environment. We first state our research questions in Section 4.1, then describe the setup used in the experiments in Section 4.2 and the datasets in Section 4.3, and conclude with the evaluation results in Section 4.4.

### 4.1. Research Questions

In this part of our study, we aim at addressing the following research questions:

1. Can the proposed algorithms poison an online-trained NN detector?
2. What is the influence of the detector's hyperparameters and attack characteristics on the poisoning effectiveness?
3. Is there any difference in the poisoning effectiveness between the synthetic signals and real-world data?
4. How do the proposed algorithms, i.e., back-gradient optimization attack and interpolation-based attack, compare against each other?
5. Can the attacks generated by the proposed algorithms be transferred to a detector with a different optimization algorithm, parameters, or architecture?

### 4.2. Evaluation Setup

This section presents the details of the 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 [29] for all tests, for both synthetic and real data. Table 1 presents one sample of such detector.

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 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 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

**Table 1**
Model summary for an UAE Detector with four input signals and sequence length of 20.

Model: UAE Detector

| Layer (type) | Output Shape | Param # |
|---|---|---|
| x (Placeholder) | (None, 20, 4) | 0 |
| flat_x (Reshape) | (None, 80) | 0 |
| encoder_inflator (Dense + tanh) | (None, 160) | 12,800 |
| encoder_0 (Dense + tanh) | (None, 53) | 8,480 |
| decoder_inflator (Dense + tanh) | (None, 160) | 8,480 |
| decoder_0 (Dense + tanh) | (None, 160) | 25,600 |
| decoder_last (Dense + tanh) | (None, 80) | 12,800 |
| out (Reshape) | (None, 20, 4) | 0 |

Total params: 68,160

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.

## 4.3. Datasets

This section describes both the synthetic and the real-world datasets used in our evaluation.

**Synthetic dataset.** For the synthetic data experiments, we used several 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 (TOP) and the lowest point of the signal (BOTTOM), as illustrated in Figure 4.3. The rationale behind testing these attack locations was to model two types of malicious signal manipulation. For the BOTTOM location, the

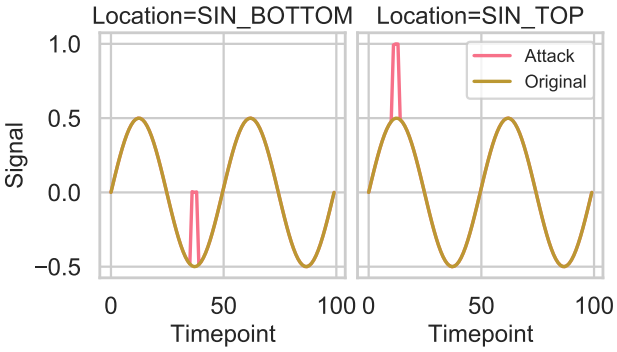


**Figure 3:** The two possible attack locations. With the BOTTOM location, the attack remains within the original signal's range; with the TOP location, the attack exceeds it.

spoofed signal stays in the range of normal signal values, while for TOP it goes beyond this range. To simulate an attacker controlling some of the sensors, the attacks were applied to just one signal.

**ICS testbed datasets.** For the real-world data experiments, we utilized three ICS datasets: SWaT [17], BATADAL [53], and WADI [1]. These datasets are very popular ICS real-world benchmark datasets, and have been used in the related research, such as: SWaT in [55, 56, 24, 33, 12, 25, 15, 44, 18, 34], BATADAL in [52, 11, 29], and WADI in [11, 29, 24].

The SWaT 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 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.

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 2. 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 many possible locations and magnitudes. In addition, in attacks #3 and #16, the attacker changes the attacked sensor's value gradually (see Figure 4.3), while in others the value changes abruptly. Thus, the selected attacks represent a variety of attack methods.

In the threat model considered, the attacker has access to the real sensory data. However, as we had no access to

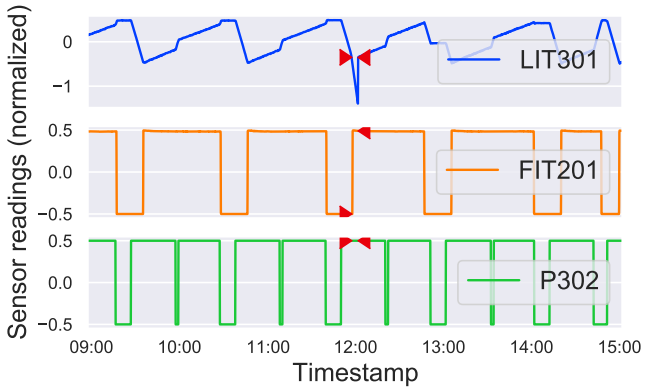


**Figure 4:** SWaT attack #16. The gradual decrease of the LIT301 water level sensor's value leads to overflow (the attack period is marked by the red arrows).

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 several 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 BATADAL dataset represents a simulated water distribution network consisting of several storage tanks, pumps, and valves, with nine PLCs controlling them. The dataset has 43 variables that represent the water tank levels, the flow and status of all pumps, and the valves pressure. The test dataset includes seven attacks that involve malicious actuator activation, PLC set point changes, and sensor measurement falsification. Two sensor manipulation attacks, 9 and 12, were used in our evaluation. Both attacks use the same sensor manipulation - reporting low water level and leading to overflow. We modeled the L_T2, F_V2, F_PU1, F_PU2 signals influenced by these attacks, with the attacker controlling the L_T2 sensor.

The WADI dataset was collected from a real-world scaled-down water distribution testbed consisting of large water tanks that supply water to smaller consumer tanks. The WADI dataset has 126 features. The 16 attacks present in the dataset aim to interfere with the water supply to the consumer tanks. The attacks involved valves opening and sensor readings spoofing, and were partially concealed by the attacker. Attack 2, where the sensor level was spoofed in order to cause an increase of chemical dosing, was used in our evaluation. The 1_MV_001_STATUS, 1_FIT_001_PV, and 1_LT_001_PV signals were modeled.

The data of all datasets was normalized to the $-0.5$ - 0.5 range.

## 4.4. Results

In this section, we present the detailed results of the poisoning attacks evaluation using both algorithms. We start with the synthetic signal poisoning results in Section 4.4.1, present the real-world benchmark datasets poisoning evaluation in Section 4.4.2, and conclude with the poisoning attacks transferability study in Section 4.4.3.

**Table 2**
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.

### *4.4.1. Synthetic signal poisoning*

To be consistent with the ratio of the attack duration to the attacked signal period for attacks in the real-world benchmark datasets, 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 a 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. 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 3 and the algorithms' execution time in Table 4.

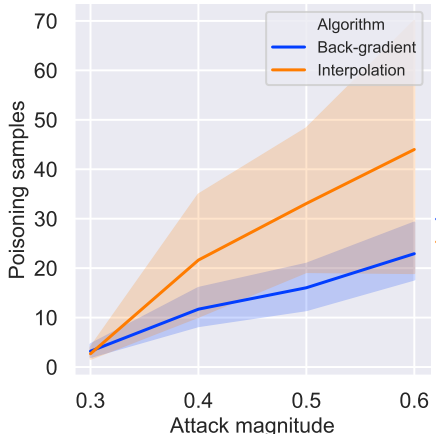


**Figure 5:** The average number of poisoning samples required to achieve the required attack magnitude for the BOTTOM location. The shaded areas show the confidence interval.

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

- **The attack magnitude**. As can be seen in Figure 5, 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 fewer poisoning samples for all tests except for those performed with the sequence length of two (see Figure 5 and Table 3).
- **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 3. 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.

**Table 3**
Comparison of maximal attack magnitude achieved by poisoning for different algorithms and locations.

| Modeled seq. 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 4**
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 5**
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 6**
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

### *4.4.2. Poisoning attacks on SWaT, BATADAL, and WADI*

For the simulated SWaT, BATADAL, and WADI attacks, we modeled a small number of the features affected by each attack (see Table 2) with a constrained attacker only able to manipulate a single sensor's data. The results are presented in Tables 5, 6, 7, and 8 with the average number of poisoning samples shown in parentheses.

**Poisoning effectiveness.** The results show that several factors influenced the poisoning's success:

- **The attack magnitude, location, and abruptness**. SWaT attacks #3 and #16, which are characterized by a gradual

**Table 7**
BATADAL attacks' 9 and 12 poisoning results

| Sequence length | Algorithm | |
|---|---|---|
| | Interpolation | Back-gradient |
| 2 | ✓(1) | ✓(1) |
| 10 | ✓(1) | ✓(1) |
| 20 | ✓(2) | ✓(40) |
| 30 | ✓(8) | ✗ |

**Table 8**
WADI attack 2 poisoning results

| Sequence length | Algorithm | |
|---|---|---|
| | Interpolation | Back-gradient |
| 2 | ✓(1) | ✓(135) |
| 10 | ✓(2) | ✓(37) |
| 20 | ✓(3) | ✓(18) |
| 30 | ✗ | ✓(32) |

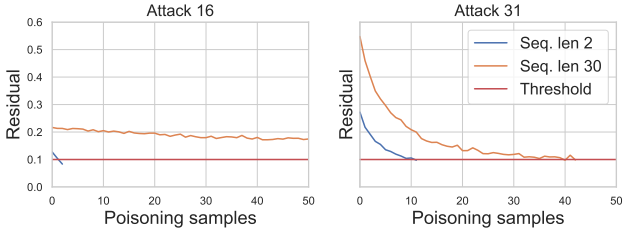


**Figure 6:** Attack detection residual change with poisoning samples addition (presented for attacks #16 and 31), and a threshold of 0.1 (the red line). With a sequence length of two, the poisoning succeeds easily, whereas the sequence length of 30 provides strong poisoning resistance and causes the poisoning to fail for attack #16.

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 the 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 resilience of the model. Increasing it caused the attacker to use more poisoning samples. Figure 6 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 SWaT 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 [28] and [29] the threshold could be as low as 0.05 without introducing false positives.

- **Algorithm**. For the SWaT, BATADAL, and WADI tests, the back-gradient algorithm did not perform better than the interpolation algorithm and required more poisoning samples in some cases (e.g. some of the WADI setups). On the other hand, the back-gradient algorithm was able to find the poisoning solution in some cases where the interpolation algorithm could not. Our analysis suggests that the difference between the synthetic and real-world datasets 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 real-world sensors (see Figure 4.3), larger steps provide better results. An optimal learning rate scheduling algorithm combining the strength of both approaches will be a topic of future research.

### *4.4.3. Transferability study*

In all experiments described above, we assumed the attacker possessed the perfect knowledge of the process and the detector. In the experiments described below, we omitted different aspects from this knowledge and explored the effects of this omission on the attack effectiveness. We conducted three types of transferability experiments that are listed in the order of the increasing difficulty for the attacker: transferring the attack (1) to a different detector optimizer, (2) to a model with different parameters, and (3) to a model of a different architecture. As during these experiments we had to test a wide range of parameters, such as sequence length and attack magnitude, the tests were performed on the synthetic data.

**Different optimizer.** In this set of the transferability tests, the detector used Adam and Momentum optimizers, while the attack poison was calculated using the gradient descent optimizer we used in all other experiments. The Momentum optimizer used the learning rate of 0.6 and the momentum of 0.5, while the Adam optimizer used the learning rate of 0.001. The parameters were selected so that the training process converged to similar error rates. We run the tests for sequence lengths between 2 and 42 and for attack magnitudes between 0.4 and 0.9 in the BOTTOM location. Each combination was run five times. The tests were conducted for the back-gradient poisoning only, as the interpolation poisoning is agnostic to the detector optimizer. The results are presented in Table 9. The results show that despite the differences between optimizers, the poisoning attacks are transferable to some extent. Out of 60 test parameters combinations, poisoning the gradient descent succeeded in 32 and was successfully transferred in thirteen cases (40%) to Momentum and in nine cases (28%) to Adam. Surprisingly, in a few cases the poison that failed with gradient descent succeeded with the alternative optimizer. This finding stresses the generality of the found poison.

**Different model parameters.** In this set of transferabil-

**Table 9**
Transferability of gradient descent (GD) back-gradient poisoning to a different optimizer. The table cells' triples represent the poisoning outcome for GD, Adam, and Momentum, respectively. ✓ denotes poisoning success; ✗ - poisoning failure.

| Seq. length | Magnitude | | | | | |
|---|---|---|---|---|---|---|
| | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| 2 | (✓, ✓, ✗) | (✗, ✓, ✓) | (✗, ✗, ✗) | (✗, ✗, ✗) | (✗, ✗, ✗) | (✗, ✗, ✗) |
| 12 | (✓, ✓, ✓) | (✓, ✗, ✓) | (✓, ✓, ✓) | (✓, ✗, ✓) | (✓, ✗, ✓) | (✓, ✗, ✗) |
| 22 | (✓, ✓, ✓) | (✗, ✓, ✗) | (✗, ✓, ✗) | (✗, ✗, ✗) | (✗, ✗, ✓) | (✗, ✗, ✗) |
| 32 | (✓, ✓, ✓) | (✓, ✓, ✓) | (✗, ✗, ✓) | (✓, ✗, ✓) | (✓, ✗, ✓) | (✓, ✗, ✓) |
| 42 | (✓, ✓, ✓) | (✓, ✓, ✓) | (✓, ✓, ✗) | (✗, ✓, ✓) | (✗, ✗, ✗) | (✗, ✗, ✗) |

ity tests, both the detector and the attacker used the same gradient descent optimizer and same autoencoder detector model. We fixed the attack magnitude at 0.4 and the location at BOTTOM. We tested sequence lengths between 2 and 42, while the detector used a *different length* from the one used by the attacker. For example, if the attacker used the length of 2, the tests with the detector's sequence lengths of 12, 22, 32, and 42 were conducted. All combinations of lengths were tested for both poisoning algorithms. The results are presented in Table 11 and in Table 12 for the interpolation and for the back-gradient poisoning respectively. The results present a very similar picture for both poisoning algorithms. In the vast majority of the test configurations, a poison that succeeded with a shorter sequence length was also successful when applied to detectors with longer sequence lengths. Contrarily, the poisons that succeeded for the attacker assuming a longer sequence failed in the vast majority of the tests for shorter sequences. This finding again suggests that the proposed algorithms' transferability. It is important to note that these tests were conducted in the *offline* setting - all poisoning samples found by the attacker were added to the detector's training set at once. As we show later in Section 6, the online training mode where the poisoning samples are tested by the detector prior to being added to the training set presents additional obstacles for successful poisoning.

**Different model architecture.** In this set of transferability tests, the attacker poisoned the autoencoder detector, while the defender used a 1D-convolution-based NN. We run the tests for sequence lengths between 2 and 42 and for attack magnitudes between 0.3 and 0.9 in the BOTTOM location for both interpolation and back-gradient algorithms. Table 10 presents the architecture of the 1D convolution detector used in the experiments. The detector's parameters were selected by grid search over hyperparameters as providing the best detection of the attacks. The 1D convolution detector was implemented using the Keras framework and trained using the same gradient descent optimizer like the one used by the attacker. The same 1D convolution detector configuration was used in all 70 attacker's test parameters combinations. In all the tests, the poisoning of the 1D convolution with the poison created for the autoencoder failed, regardless of the poisoning result on the attacker's autoencoder. To get a more precise picture of the poisoning effect we compared the decrease of the detector's residual due to poisoning by subtracting the residual produced by the poisoned model from the residual of the unpoisoned model. As shown in Figure 7, the residual decrease of the (unknown to the attacker) defender's model is comparable with the decrease of the attacker's model. This is not enough to conceal the attack but demonstrates the generality of the proposed algorithms.

**Table 10**
Model summary for 1D Conv Detector.

Model: 1D Conv Detector

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_1 (InputLayer) | (None, 18, 2) | 0 |
| conv_0 (Conv1D) | (None, 17, 16) | 80 |
| max_pooling1d_1 (MaxPooling1D) | (None, 8, 16) | 0 |
| conv_1 (Conv1D) | (None, 7, 32) | 1056 |
| max_pooling1d_2 (MaxPooling1D) | (None, 3, 32) | 0 |
| conv_2 (Conv1D) | (None, 2, 64) | 4160 |
| max_pooling1d_3 (MaxPooling1D) | (None, 1, 64) | 0 |
| flat_last (Flatten) | (None, 64) | 0 |
| dense_last (Dense) | (None, 2) | 130 |
| reshape_1 (Reshape) | (None, 1, 2) | 0 |

Total params: 5,426
Trainable params: 5,426
Non-trainable params: 0

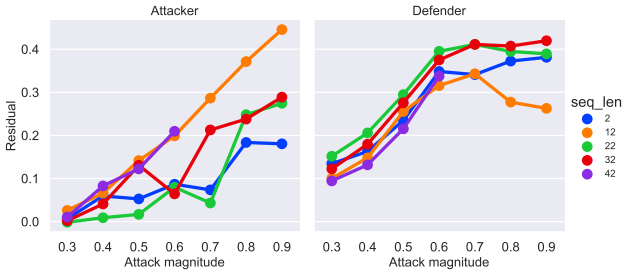


**Figure 7:** Attack detection residual for the attacker's and the defender's models for back-gradient poisoning. The attacker used autoencoder, while the defender - 1D convolution. The defender's model exhibits a comparable residual decrease.

## 5. Tennessee Eastman Process Simulator

### 5.1. Research Questions

The synthetic data and the static data collected from the SWaT testbed provide a good starting point for assessing the effectiveness of the proposed poisoning algorithms. How-

**Table 11**
Transferability of the interpolation poisoning to a model with different sequence lengths. The attacker's sequence lengths appear in the rows, the detector's - in the columns. ✓ denotes poisoning success; ✗ - poisoning failure.

| Attacker's seq. length | Attacker's outcome | Detector sequence length | | | | |
|---|---|---|---|---|---|---|
| | | 2 | 12 | 22 | 32 | 42 |
| 2 | ✓ | - | ✗ | ✓ | ✓ | ✓ |
| 12 | ✓ | ✓ | - | ✓ | ✓ | ✓ |
| 22 | ✓ | ✗ | ✗ | - | ✓ | ✓ |
| 32 | ✓ | ✗ | ✗ | ✗ | - | ✓ |
| 42 | ✗ | ✗ | ✗ | ✓ | ✓ | - |

**Table 12**
Transferability of the back-gradient poisoning to a model with a different sequence length. The attacker's sequence lengths appear in the rows, the detector's - in the columns.✓ denotes poisoning success; ✗ - poisoning failure.

| Attacker's seq. length | Attacker's outcome | Detector sequence length | | | | |
|---|---|---|---|---|---|---|
| | | 2 | 12 | 22 | 32 | 42 |
| 2 | ✓ | - | ✗ | ✓ | ✓ | ✓ |
| 12 | ✓ | ✗ | - | ✓ | ✓ | ✓ |
| 22 | ✓ | ✓ | ✓ | - | ✓ | ✓ |
| 32 | ✓ | ✗ | ✗ | ✗ | - | ✓ |
| 42 | ✓ | ✗ | ✗ | ✗ | ✗ | - |

ever, they are not enough for evaluating the algorithm's performance in dynamic systems. In this part of our study, we aim at addressing the following research questions:

1. Can the proposed poisoning algorithms be applied to a dynamically controlled system and what changes are required to apply them?
2. What is the effectiveness of the proposed algorithms when applied to a dynamic system?
3. Can poison be generated for different kinds of attacks on dynamic systems?
4. How do the proposed algorithms compare against each other?

The most realistic evaluation environment would include a testbed implementing some industrial process, an NN-based detector monitoring it, and a compromised sensor used to carry out the attack. Unfortunately, using such a real-world testbed for our experiments would be very costly, as hundreds and thousands of experiments are required for a thorough study of the complex interaction between the attacker, the process, and the detector. Moreover, attacking a real-world testbed could present safety risks as the attacks might lead to such effects as high pressure, overheating, or overflow.

In order to test the algorithms in a realistic, but safe, controlled, and repeatable way we used a simulated Tennessee Eastman (TE) industrial process [10]. The TE process model is a simulation of a chemical plant, which became a standard

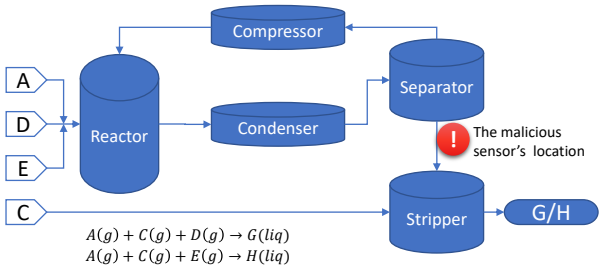


**Figure 8:** A schematic TE process diagram.

benchmark in the field of control and monitoring [54], and is widely used for data-driven fault detection. It has been used as a base for several cybersecurity testbeds such as the NIST's cybersecurity performance testbed [7], Damn Vulnerable Chemical Process [31], and GRFICS [13]. Consequently, many recent cyberattack and anomaly detection studies [26, 14, 35, 21, 41] use TE as a test case.

## 5.2. TE Process Description

The TE process is an open-loop model representing a complex non-linear real-world chemical process, described in detail in [10] and is presented schematically in Figure 8. The process produces two liquid products, $G$, and $H$, from four gaseous reactants $A$, $C$, $D$ and $E$. The process involves five major components: a reactor, a condenser, a vapor-liquid separator, a product stripper, and a compressor. The reactants are mixed in the reactor in an irreversible exothermic reaction. The products leave the reactor and are cooled down in the condenser. Non-condensed gases are recycled by the compressor back to the reactor. The condensed stream is processed by the separator and, finally, the stripper removes the remaining reactants from the final liquid product. The process can operate in six different modes that reflect the desired values of the $G/H$ ratio as well as of the production rate; we used the base case - Mode 1. The process has 41 measurements, XMEAS(1) through XMEAS(41), and 12 manipulated variables, XMV(1) through XMV(12). For example, XMEAS(1) measures the feed of component A, and XMV(3) allows controlling this feed by manipulating the corresponding valve. There is no mathematical model of the process, and the process state is calculated by the code provided which is intentionally obfuscated. The process must be controlled to meet the objectives that include: (1) maintaining the desired process variables, (2) keeping the process within the equipment safety constraints, (3) recovering quickly from disturbances. This last objective is especially relevant for us, as it directly interferes with the attacker's objective. There are five constraints that ensure safe equipment operation. The constraints impose both low and high limits on the allowed reactor's pressure, level, and temperature, separator level, stripper base level. Violating any constraint causes the process shutdown. The original TE benchmark includes twenty process disturbances, such as component $A$ feed loss. The disturbances are used as tests for comparing

control strategies.

In our experiments, we used the revised Simulink simulation of the TE process of Bathelt *et al.* [3]. The simulation solves the control problem with the help of twenty *proportional-integral* (PI) controllers monitoring the XMEASs and influencing the process by changing the XMVs.

### 5.3. TE Process Threat Model and Attacks

In this experiment, we used the threat model described in Section 2—an attacker possessing complete knowledge about the attacked system and capable of falsifying a sensor's reading at the desired time. As in prior experiments, we limited the attacker's tampering ability to a single sensor; adding more sensors increases the attacker's power. There are multiple possible attack goals, for example, decreasing operational efficiency of the plant [21] or causing a shutdown in the shortest amount of time [30]. In our study, the attacker's goal is to damage the system's availability and incur operational costs by forcing the system to shut down. In contrast to an attack aimed at finding the shortest way to shut the process down [30], we focus on stealthy attacks that (1) cause a shutdown in a location that is different from the malicious sensor's location, and (2) delay the shutdown until after the tampering of the sensor's reading has been completed. An example of such stealthy attacks is the infamous Stuxnet malware [32] that targeted nuclear centrifuges in Iran. To the best of our knowledge, no prior work has presented such attacks on the TE process. Therefore, we devised several novel attacks as a part of this research, as described below.

As the TE process does not have a mathematical model, finding such stealthy attacks was an empirical experimental process. This attack exploration turned out to be a difficult task due to the inherent robustness of the control part of the TE simulator. As mentioned above, the TE process model already contains a number of process disturbances, and the control strategies are tested against these disturbances. Hence, the controllers successfully cope with small and medium changes of the measurements' values, and the process stays within the safety limits. On the other hand, large changes in a sensor reading immediately trigger the safety alarms, and the system is shut down abruptly and non-stealthily. In order to be able to conduct the attacks and study their effects, we augmented the simulation with the MATLAB function that intercepts the sensors' readings between the process block and the PI controllers and enables their manipulation (see Figure 9). The function (indicated by the red exclamation mark in Figure 9) hosts the code that analyzes the current process state reported by the sensors, calculates the poison or attack value to override the sensor under attack, and selects the right moment to replace the measured value with the calculated one with the help of the shutdown predictor (described in Section 5.4). The attacker's code is concentrated in a single location, thus making this attack generation method generic and applicable to other Simulink-based process simulations. The Simulink/MATLAB models and the attack generation code are open source and freely available on GitHub.[2]

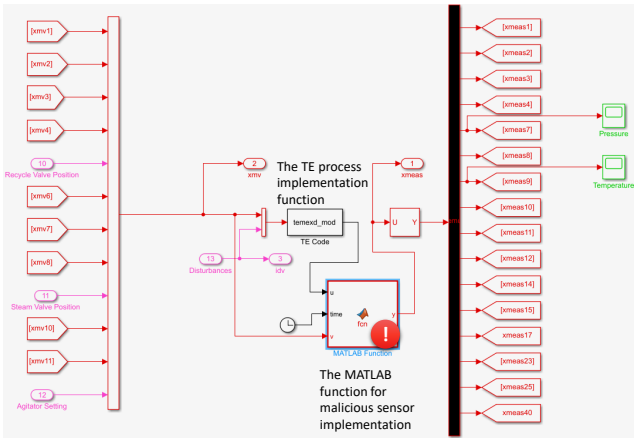


**Figure 9:** Attacking the simulated TE process. The Simulink model of the TE process is augmented with the MATLAB code for conducting the experiments and attacks. The MATLAB function hosts all the code for reading the sensors, calculating the poison and the attack, and executing both.

Several factors increased the difficulty of identifying both effective and stealthy attacks. The high nonlinearity of the process, active changes by the controllers to restabilize the process once a disturbance is introduced, and complex inter-controller dependency made a prediction of the sensor manipulation's precise effect close to impossible. Moreover, as the process has no mathematical model, there is no way to calculate the required attack working back from the desired outcome. We explored two kinds of attacks: (1) changing the real sensor value by a given amount, and (2) setting the sensor value to a given value. These two kinds of attacks represent a miscalibrated sensor and a stuck sensor, respectively. While there are many possible attack points, we limited our search to the incoming flow of component A and the separator underflow, where we performed most of our attacks. After an extensive search of possible attack spaces, we selected four different attacks on the TE process which have not been described in prior work. The attacks are described in Table 13. As Table 13 shows, the attacks differ from each other in magnitude, duration, and way of altering the signal. In order to find an attack, we selected the kind (relative or fixed change) and run extensive tests over multiple values of the magnitude and duration until the minimal values that still cause the attack were found. The smallest possible attack is used in order to prevent a premature shutdown by the poison that gets incrementally closer to the attack. To understand the complexity of attack crafting, consider Figure 10 that illustrates some of the processes taking place during TE attack #2. The attack increases the separator underflow sensor value by 7. This raise causes the separator flow rate controller to close the valve which compensates for the flow raise. Closing the valve causes the flow to decrease (even though the attack still reports much higher values) and the

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

**Table 13**
TE attack description.

| # | Attacked sensor [magnitude] | Modeled sensors | Description | Impact |
|---|---|---|---|---|
| 1 | XMEAS(1) [+2.35] | XMEAS(1), XMEAS(7), XMEAS(8), XMV(3) | Increase the real value by 2.35 for 3h | High reactor pressure or low stripper level shutdown |
| 2 | XMEAS(14) [+7] | XMEAS(14), XMV(7), XMEAS(15), XMV(8) | Increase the real value by 7 for 2.88h | High stripper level shutdown |
| 3 | XMEAS(14) [-7] | XMEAS(14), XMV(7), XMEAS(15), XMV(8) | Decrease the real value by 7 for 2.02h | Low separator level shutdown |
| 4 | XMEAS(14) [22.9] | XMEAS(14), XMV(7), XMEAS(12) | Set the value to 22.9 for 1.9h | Low separator level shutdown |

Legend: XMEAS(1) - A feed, XMEAS(7) - Reactor pressure, XMEAS(8) - reactor level, XMEAS(12) - separator level, XMEAS(14) - separator underflow, XMEAS(15) - stripper level, XMV(3) - A feed flow, XMV(7) - separator flow, XMV(8) - stripper flow.

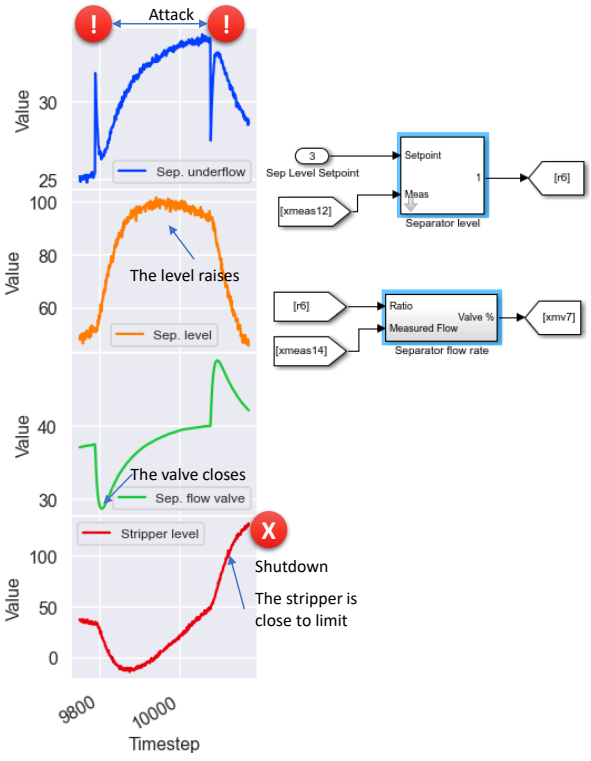


**Figure 10:** Attack 2 of the simulated TE process. The graphs presents the values of XMEAS(14), XMEAS(12), XMV(8), XMEAS(15) (top to bottom). Adding a fixed value to the real value of XMEAS(14) eventually causes XMEAS(15) to reach the dangerous high limit and triggers the safety shutdown.

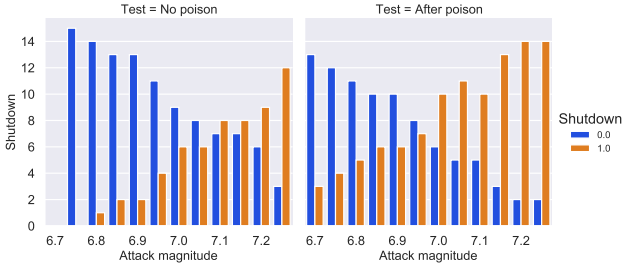


**Figure 11:** Histogram of shutdowns during TE Attack 3 with and without poisoning. The attacks were conducted using a number of magnitudes, shown at the x-axis. Each attack magnitude was tested fifteen times with different delays. The left histogram presents the test with attacks conducted without prior poisoning. The right side - attacks after poisoning.

level starts to rise. The rising level causes the valve to gradually open, eventually reaching a steady state. At the same time, the stripper level increases, because the flow from the separator fills it up. When the attack finishes, the sensor reports the real value of the separator underflow, which is much lower than the previously reported. This causes the valve to open and a large inflow from the separator into the stripper, causing it to reach an unsafe state and shut down. This description is partial as we omitted the interaction with the controllers regulating the stripper level and flow rate and the influence of the attack on the rest of the system. The simulated TE process presented two new substantial challenges to the proposed poisoning methods, that were absent from the static data: (1) the shutdown predictability and (2) precise modeling of the system's response to the poison.

**The shutdown predictability.** The attack timing plays a crucial role in its outcome. The same attack carried out at a slightly different time might have a different outcome, depending on the exact state of the entire system, as illustrated in Figure 11. Figure 11 presents the histogram of the attack 3 outcomes while the attack was performed using different attack magnitudes with fifteen increasing time delays between 0 and 7.5 hours from the chosen attack time. On the left side, the attacks were carried out after 63 hours of the process run, on the right side - after the same time, but with four poisons introduced during this time period. Two observations can be made based on this histogram. First, the same attack magnitude could cause shutdown or not cause it - depending on the attack's timing. Second, after the poisoning, the system had more shutdowns at every attack magnitude and these shutdowns started at lower magnitudes than without poisoning. Consequently, applying poisons could trigger a premature shutdown, which the attacker usually wants to avoid. With other attacks, the attack outcome was even less predictable.

**The system response to the poison.** As shown in Figure 10,

the controllers react to the introduced poison and cause abrupt changes in many signals. In order to avoid triggering the detector's alarm in these dependent signals, a way to precisely predict the system's response while calculating the poison is required. As illustrated in Figures 10 and 11, the system reaction to the disturbance is complex and dependent on many variables of the system state, thus simple interpolation using the system response to the attack was not enough. Failing to predict the system's response caused the detector to trigger alerts on the poison injected into the simulator. We describe our solution to these two challenges below.

## 5.4. TE Simulator Poisoning Setup

In order to cope with the new challenges of the dynamic system, we augmented our attack with two new components: (1) process predictor (PP) and (2) shutdown predictor (SP). The goal of the PP is to predict the values of the variables monitored by the detector as a result of the injected poison. In theory, we could find the precise system response by integrating the simulator into the poison calculation. However, in practice this approach would suffer from very serious performance issues as running the simulator for each poison calculation iteration would make it very slow. Therefore, we replace the simulator with the PP that approximates it, can be naturally integrated into the algorithm, and provides excellent performance. The PP is integrated into the poison calculation process as follows. The algorithm described in Section 3.2.1 and Section 3.2.2 work as described with a single change: after calculating the next value of the poison we use PP to predict the values of the dependent signals from the signal under attack. The architecture of the PP is presented in Table 14. The PP model predicts the next values of all modeled sensors based on the previous values of the signal which is being poisoned (see Table 13). For example, for attack #2 it predicts XMV(7), XMEAS(15), XMV(8) based on XMEAS(14). The model has three fully connected layers followed by a dropout with a rate of 0.2 and a final dense layer producing the output. In addition to the attacked sensor signal (which is fully known to the attacker), it has an auxiliary input, which we called the *bootstrapping* input. The bootstrapping input, which is very short compared to the length of the regular input, contains the values of all modeled signals as they are under the normal operating conditions. Unlike the values of the real signals which are unknown when the attack or poison is injected, the values under normal conditions are known in advance. We found that the bootstrapping input increased the PP's accuracy in predicting the process behavior. The reason for that improvement is that bootstrapping allows the model to establish a baseline for the predicted signals in relation to the input (i.e. manipulated) signal. The predictor was implemented in Keras and used 300 last values of the attacked signal and the bootstrapping input of length 50. The predictor was trained on the data collected from the simulator while conducting attack-like signal manipulation in the signal range around the target attack magnitude, with a third of the data set aside as validation and tested on the simulator's reaction to the generated poison. A dedicated predictor was created for each attack.

**Table 14**
Model summary for TE Process Predictor.

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 400, 1) | 0 | |
| bootstrapper (InputLayer) | (None, 50, 4) | 0 | |
| flat (Flatten) | (None, 400) | 0 | input_1[0][0] |
| bootstr_flat (Flatten) | (None, 200) | 0 | bootstrapper[0][0] |
| concat_1 (Concatenate) | (None, 600) | 0 | flat[0][0]<br>bootstr_flat[0][0] |
| dense_0 (Dense) | (None, 900) | 540900 | concat_1[0][0] |
| dense_1 (Dense) | (None, 225) | 202725 | dense_0[0][0] |
| dense_2 (Dense) | (None, 900) | 203400 | dense_1[0][0] |
| dropout_1 (Dropout) | (None, 900) | 0 | dense_2[0][0] |
| dense_last (Dense) | (None, 3) | 2703 | dropout_1[0][0] |
| reshape_1 (Reshape) | (None, 1, 3) | 0 | dense_last[0][0] |

The goal of the SP is to predict whether the system will shut down in response to the given poison or attack. This is a classification problem that was solved using a RandomForestClassifier model with 100 estimators from sklearn Python package, which showed the best $F1$ score when compared to LogisticRegression, LogisticRegressionCV, and a fully connected NN classifiers (see Table 16). The SP was trained with the data collected from the simulator while conducting attacks with magnitude and duration in the range around the values of these variables for the specific attack. Commonly, the range of magnitudes started 1.5-2 unites below the attack magnitudes and finished at the attack magnitude and the range of duration span around one-tenth of an hour with the attack duration in the middle. The SP was integrated into the attack process as follows. Before injecting a poison or an attack, the probability of the shutdown was estimated using the trained SP. Poison was not injected until the probability was below a threshold, typically set at 0.4. The attack was not carried out unless its probability is above 0.8.

Figure12 summarizes the final setup of the TE simulator poisoning. The process consisted of the following phases:

1. **Detector training**. The detector was trained on the normal simulation data. We found it useful to add light disturbances into the training data to increase the training set variability.

2. **Attack engineering**. Once the target attack scenario is selected the simulator was used to establish the suc-

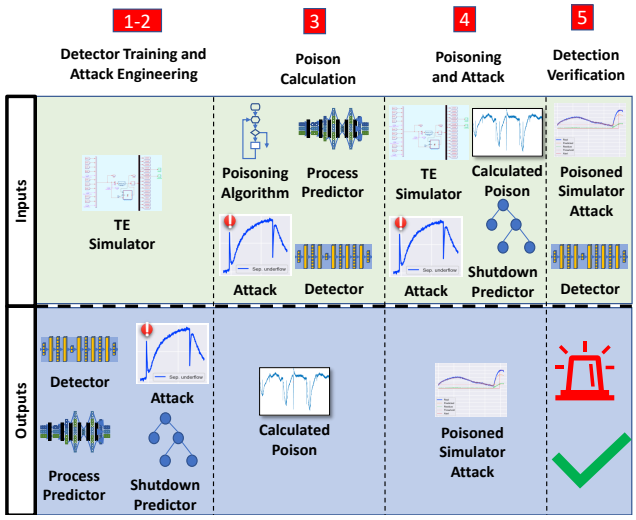


**Figure 12:** The five phases of the TE simulation poisoning method.

cessful attack conditions: the magnitude, the duration, the poison length, and the delay of the attack start relative to the signal period. The poison length includes the time of the attack itself as well as the no attack period before and after the attack allowing the system to recover from the poisons. Without this recovery period, the poison could cause a premature shutdown. The outcomes of this phase were: (1) the attack data values of all relevant signals (2) trained PP and SP models;

3. **Poison calculation**. We iteratively calculated the next poison candidates using the algorithms presented in Section 3.2. After calculating the poison values for the attacked signal, we used the PP to calculate the values of all other modeled signals before testing them with the detector. The outcome of this phase was a sequence of the found poisoning samples.

4. **Poisoning and Attack Injection**. The found poisoning samples were injected sequentially into the running simulation process. The SP was used to find the safe time for the poison injection and for the appropriate time for conducting the attack. The outcome of this phase was the recording of the system behavior under the injected poisons and attack.

5. **Attack Detection Verification**. The recorded signals containing the actual system behavior under poisoning and attack were passed through the detector. First, the poison part is used to train it in an online way, then the attack is tested.

The desired outcomes of the entire process were (1) the poison not triggering alerts, (2) the poison not causing the shutdown, (3) the attack succeeding, and (4) the attack being not detected or being detected at the very end immediately before the shutdown. We present the results of these experiments in Section 5.5.

## 5.5. Poisoning Tennessee Eastman Process Static Data

In the first set of experiments, we conducted poisoning of the TE process signals in the same setup used for the SWaT dataset. The goal of these experiments was to compare the TE process to the SWaT process poisoning findings. For that purpose, we created two series of attacks based on data manipulation only (not involving the actual simulation). The two sets were based on the signals used for the TE attacks 2 and 3 (see Table 13. These two attacks represent changing the signal towards higher (attack 2) and lower (attack 3) values relative to its real value and come to represent a generic family of such attacks. The original signals were scaled to be between -0.5 and 0.5, as in the synthetic data experiments described in Section 4.3. For each attack, we tested the attack magnitudes (the relative signal distortion) up to 0.5 for attack 2 and -0.5 for attack 3. We tested the same sequence lengths as in all other experiments and run each configuration at least 3 times. The results are presented in Table 15.

**Poisoning effectiveness.** In general, both algorithms successfully generated poisoning for the target attacks for the majority of test configurations. The following observations can be made:

- **Interpolation algorithm's stability**. The interpolation poisoning demonstrated stable and predictable results conforming to the findings of the experiments with the synthetic and SWaT data. Larger attack magnitudes required more poisoning samples and longer sequences were harder to poison.

- **Back-gradient algorithm's lack of predictability**. On the other hand, the back-gradient algorithm's results had a high variance in the required number of poisoning samples and these numbers were higher than of the interpolation algorithm. At the same time, it has succeeded to create a poison where the interpolation algorithm has failed (e.g. attack 2, sequence length 42). Our analysis indicates that the high samples number's variance is most probably caused by the very noisy gradients, which are due to the abrupt changes in the signals.

## 5.6. Poisoning Tennessee Eastman Process Simulator

In the second set of TE experiments, we conducted poisoning of the TE process simulator. We followed the five-stage process described in Section 5.4. The most difficult aspect of the simulator's poisoning appeared to be the influence of the poison on the system behavior and on the attack outcome. Adding a poison and the introduced process state change, causing the controllers to react and further alter the process state, thus influencing its response to the subsequent poisons and to the attack. So in order to engineer the attack, we needed to anticipate the reaction to the poison, and in order to calculate the poison, we needed to know the form

**Table 15**
Poisoning attacks on static TE process signals. ✓ denotes poisoning success; ✗ - poisoning failure. The median number of the required poisoning samples is shown in parenthesis.

| Attack # | Algorithm | Seq. length | Attack magnitude | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 0.05 | 0.10 | 0.20 | 0.30 | 0.40 | 0.50 |
| 2 | Back-gradient | 2 | ✓(36) | ✓(34) | ✓(55) | ✓(18) | ✓(10) | ✓(11) |
| | | 12 | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) |
| | | 22 | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) |
| | | 32 | ✓(1) | ✓(68) | ✓(21) | ✓(18) | ✓(45) | ✓(157) |
| | | 42 | ✓(56) | ✓(87) | ✓(45) | ✓(50) | ✓(114) | ✓(46) |
| | Interpolation | 2 | ✓(1) | ✓(2) | ✓(2) | ✓(2) | ✓(2) | ✓(3) |
| | | 12 | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) |
| | | 22 | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) |
| | | 32 | ✓(1) | ✓(2) | ✓(7) | ✓(10) | ✓(12) | ✓(15) |
| | | 42 | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| 3 | Back-gradient | 2 | ✓(1) | ✓(13) | ✓(14) | ✓(14) | ✓(10) | ✓(11) |
| | | 12 | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) |
| | | 22 | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(50) |
| | | 32 | ✓(1) | ✓(29) | ✓(38) | ✓(138) | ✗ | ✗ |
| | | 42 | ✗ | ✓(26) | ✓(69) | ✗ | ✗ | ✗ |
| | Interpolation | 2 | ✓(1) | ✓(2) | ✓(2) | ✓(2) | ✓(2) | ✓(2) |
| | | 12 | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) |
| | | 22 | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(1) | ✓(2) |
| | | 32 | ✓(1) | ✓(12) | ✓(18) | ✓(22) | ✓(24) | ✓(29) |
| | | 42 | ✓(19) | ✓(48) | ✓(70) | ✓(101) | ✗ | ✗ |

**Table 16**
$F1$ scores of shutdown predictors for each attack.

| Attack # | Test cases num. | LR | LRCV | NN | RF |
|---|---|---|---|---|---|
| 1 | 19537 | 0.633 | 0.633 | 0.901 | 0.959 |
| 2 | 518 | 0.865 | 0.865 | 0.970 | 1 |
| 3 | 966 | 0.760 | 0.740 | 0.902 | 0.938 |
| 4 | 5035 | 0.684 | 0.723 | 0.719 | 0.75 |

LR - Logistic Regression, LRCV - Logistic Regression CV, RF - Random Forest, NN - Neural Network.

of the attack. Performing a systematic grid search over the attack lengths and magnitudes and training both the SP and the PP with data collected over a wide range of scenarios simulating different poison-like disturbances allowed us to successfully poison the simulator in all four TE attacks scenarios. Table 16 shows the $F1$ classification metric for different shutdown predictor models. The NN SP model was a fully connected 3-layers network with 64 units, ReLU activations, and Dropout layers between the Dense layers with the rate of 0.25. It was implemented in Keras and trained using Adam optimizer with binary cross-entropy as a loss function. All classifiers used balanced class weights and 20-80 test-train split. Random Forest classifier showed the best performance and was used in all the attacks.

All tests were conducted with the detector using the sequence length of 20. The back-gradient algorithm's lack of predictability on the TE signals already demonstrated on the static data experiment was ever larger in the real simulator experiment. This lack of predictability prevented us from generating suitable data for training the process and shutdown predictor. In addition, the response of the system to the poison introduced even more noise into the gradients and made the back-gradient convergence very slow. For these reasons, we limited our experiments to a more stable and predictable interpolative algorithm. Table 17 summarizes the poisoning of the four TE attacks on the simulator and Figures 13 and 14 illustrate a successful poisoning and attacking in the TE attack 2 setup. Figure 13 shows all four signals of the simulator while the poisons and the attack are injected into XMEAS(14), shown at the top. It can be seen that the attacker waits for a long time before launching the attack, waiting for the shutdown predictor to indicate the correct moment for the attack. Also, it can be seen that while the attack is conducted on XMEAS(14), the shutdown occurs due to the safety limit violation on XMEAS(15) (the third signal from the top). Figure 14 shows the detector's state for the XMEAS(15) signal without poisoning (on the left) and after poisoning (on the right). The poisoning causes the predicted value to get closer to the observed one thus removing the alert. Only immediately before the shutdown, the detector starts triggering an alert, but this happens after the attack has completed. The collected datasets for the SP and PP, trained models, the attack signals, and the detector's results for the attacks are freely available.[3].

## 6. Poisoning Attack Mitigation

Our findings suggest that **decreasing the detection threshold** is an effective means of poisoning mitigation. Figure 15

[3]Link will be provided later

**Table 17**
Poisoning attacks on TE process simulator. ✓ denotes poisoning success; The number of the required poisoning samples is shown in parenthesis.

| Attack # | Alerts without poison | Alerts with poison | Poisoning outcome (samples) |
|---|---|---|---|
| 1 | 276 | 0 | ✓(11) |
| 2 | 349 | 7[1] | ✓(5) |
| 3 | 73 | 0 | ✓(4) |
| 4 | 133 | 0 | ✓(4) |

[1] The alerts immediately preceded the shutdown thus providing no early notice to the defender.

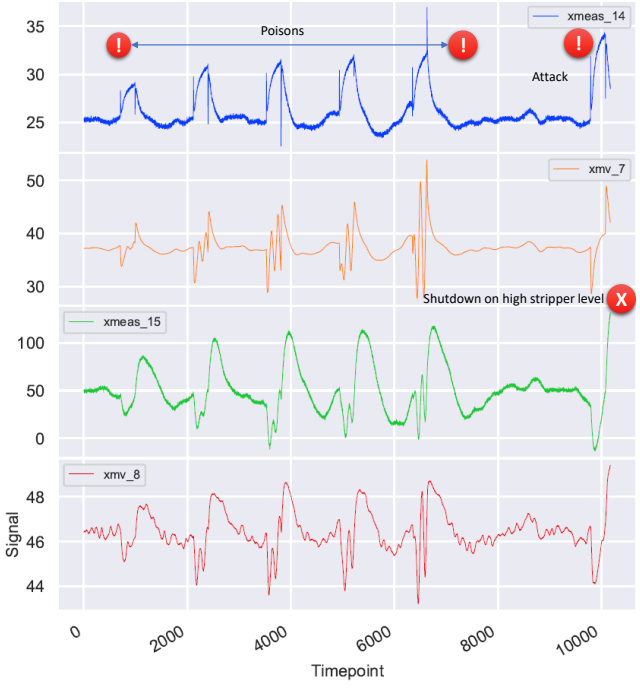


**Figure 13:** Attack 2 for the TE process simulator. The picture shows the five poisons followed by the attack after a prolonged waiting until the SP predicts a certain shutdown, which indeed happens as the value of XMEAS(15) becomes too high.

illustrates this effect for attack #7, the same behavior 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 5 and 6). 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 18, increasing the length results in just a 2-4% decrease in the detection score.

As the 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

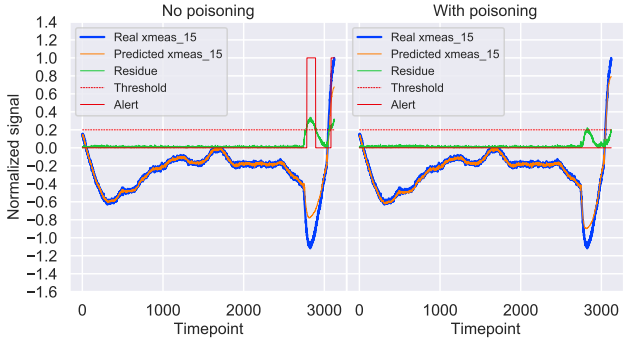


**Figure 14:** The poisoning effect during Attack 2 for the TE process simulator. On the left, where the attack is conducted without poisoning the residual between the real and predicted signal is above the threshold and an alert is issued at the beginning of the attack. On the right side, the residual is kept below the threshold due to prior poisoning and an alert happens after the attack has ended, right before the shutdown.

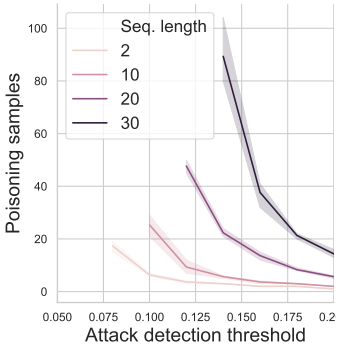


**Figure 15:** The average number of poisoning samples required to achieve the SWaT attack #7 under the given detection threshold. Lower thresholds require more points. It was not possible to achieve the attack for thresholds lower than the left-most point of each graph.

**Table 18**
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 |

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 19, 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 19**
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) |

## 7. Related Work

Several recent studies have focused on evasion attacks on CPS anomaly detectors. In [12], 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 [14] 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.* [11] demonstrated a successful real-time evasion attack on an autoencoder-based detection mechanism in water distribution systems. The authors of [11] 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. Another recent paper in this area [56] 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. In [33], a framework for generating adversarial examples generation for CPS with linear input constraints is presented and successfully evaluated with simulated data. In [24], the authors present a successful evasion of both NN detector and additional rules checkers in CPSs with the help of genetic algorithms evaluated on two public datasets. Our study differs from these studies in several ways. First and foremost, all of the abovementioned papers examined evasion attacks, while our research focuses on poisoning attacks. Second, [12, 11, 56] 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.* [39] 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.* [48] and Suciu *et al.* [50] 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.* [37] 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. Table 20 summarizes related studies and compares them to our research.

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.

## 8. 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 has been presented. We also demonstrated that attacks are transferrable to detectors different from those used to generate the attacks, albeit in a limited form. Lastly, we presented a method for applying poisoning attacks to dynamic systems and successfully conducted four and novel attacks on the TE process simulator.

Our results also point out several factors influencing the examined detectors' robustness to poisoning attacks. First, in most 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 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 causing 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.

**Table 20**
Comparison of related adversarial attacks studies.

| Ref. | Application area | Attack location | Attacked detector model | Online | Attack target | Attack type |
|---|---|---|---|---|---|---|
| [12] | ICS IDS | Sensor + Detector | Recurrent NN (LSTM) | No | Static | Evasion |
| [14] | ICS IDS | Sensor + Detector | Feedforward NN | No | Dynamic | Evasion |
| [37] | Network IDS | IDS | Autoencoder NN | Yes | Static | Poisoning |
| [11] | ICS IDS | Sensor + Detector | Autoencoder NN | No | Static, Dynamic | Evasion |
| [56] | ICS IDS | Detector + IDS | Recurrent NN (LSTM) | No | Static | Evasion |
| [33] | ICS IDS | Sensor only | Feedforward NN | No | Static | Evasion |
| [24] | ICS IDS | Sensor only | Recurrent NN (LSTM) | No | Static | Evasion |
| **Ours** | ICS IDS | Sensor only | Autoencoder NN | Yes | Static, Dynamic | Poisoning |

Our experiments show that neither of the algorithms has a definitive advantage when evaluated with real-world datasets. The back-gradient algorithm required more poisoning points in some cases, yet it was able to find a solution in several instances where the interpolation algorithm failed. The interpolation algorithm generally performed better when the detector used shorter sequences, while the back-gradient performed better with longer sequences. Therefore, the algorithms will complement each other in the attacker's toolbox. As both the interpolation and back-gradient algorithms are contributions of this research, we believe that presenting the algorithms' advantages and limitations has scientific value. In addition, TE simulator experiments have revealed another difficult challenge for the attacker: predicting the poisoning's effect on a dynamic system. The dynamic reaction of the control loops and of the controlled process significantly complicates the attacker's ability to manipulate the process and the detector. The TE process's control strategy may pose a particularly difficult challenge to the attacker [30]. Follow-up research with other real-world dynamic systems is required to assess this aspect of ICS poisoning susceptibility. One of the main goals of this research was to investigate whether adversarial poisoning attacks that were evaluated on static datasets perform similarly in dynamic environments. Our results show that this is not the case, suggesting that such attacks must be evaluated in a dynamic environment to understand their actual impact.

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 mostly focused on autoencoder-based detectors; studying the robustness of other NN architectures is a topic for future research. 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. We have discovered that real-world dynamic systems present multiple challenges for the poisoning attacker. The two most important challenges to be researched are: (1) improving the attacker's ability to engineer the attack while accounting for the system response to the poisoning and (2) adapting the back-gradient optimization for dynamic systems attacks.

## 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 Israel-U.S. Binational Industrial Research and Development (BIRD) Foundation; by the PRIN 2017 project RexLearn (grant no. 2017 TWNMH2), 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] Ahmed, C.M., Palleti, V.R., Mathur, A.P., 2017. Wadi: A water distribution testbed for research in the design of secure cyber physical systems, in: Proc. 3rd Int' Workshop on Cyber-Physical Systems for Smart Water Networks, ACM. pp. 25–28.

[2] Ahmed, C.M., Zhou, J., Mathur, A.P., 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, pp. 566–581.

[3] Bathelt, A., Ricker, N.L., Jelali, M., 2015. Revision of the tennessee eastman process model. IFAC-PapersOnLine 48, 309–314.

[4] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F., 2013. Evasion attacks against machine learning at test time, in: Joint European conference on machine learning and knowledge discovery in databases, Springer. pp. 387–402.

[5] Biggio, B., Roli, F., 2018. Wild patterns: Ten years after the rise of adversarial machine learning. Pattern Recognition 84, 317–331.

[6] Bitton, R., Maman, N., Singh, I., Momiyama, S., Elovici, Y., Shabtai, A., 2021. Evaluating the cybersecurity risk of real world, machine learning production systems. arXiv preprint arXiv:2107.01806 .

[7] Candell, R., Zimmerman, T., Stouffer, K., et al., 2015. An industrial control system cybersecurity performance testbed. National Institute of Standards and Technology. NISTIR 8089.

[8] Demetrio, L., Biggio, B., Lagorio, G., Roli, F., Armando, A., 2021a. Functionality-preserving black-box optimization of adversarial windows malware. IEEE Transactions on Information Forensics and Security 16, 3469–3478.

[9] Demetrio, L., Coull, S.E., Biggio, B., Lagorio, G., Armando, A., Roli, F., 2021b. Adversarial EXEmples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection. ACM Trans. Priv. Secur. 24.

[10] Downs, J.J., Vogel, E.F., 1993. A plant-wide industrial process control problem. Computers & chemical engineering 17, 245–255.

[11] Erba, A., Taormina, R., Galelli, S., Pogliani, M., Carminati, M., Zanero, S., Tippenhauer, N.O., 2020. Constrained concealment attacks against reconstruction-based anomaly detectors in industrial control systems, in: Annual Computer Security Applications Conference, pp. 480–495.

[12] Feng, C., Li, T., Zhu, Z., Chana, D., 2017. A deep learning-based

framework for conducting stealthy attacks in industrial control systems. arXiv preprint arXiv:1709.06397 .

[13] Formby, D., Rad, M., Beyah, R., 2018. Lowering the barriers to industrial control system security with {GRFICS}, in: 2018 {USENIX} Workshop on Advances in Security Education ({ASE} 18).

[14] Ghafouri, A., Vorobeychik, Y., Koutsoukos, X., 2018. Adversarial regression for detecting attacks in cyber-physical systems, in: Proc. 27th Int'l Joint Conf. Artificial Intelligence, pp. 3769–3775.

[15] Giraldo, J., Sarkar, E., Cardenas, A.A., Maniatakos, M., Kantarcioglu, M., 2017. Security and privacy in cyber-physical systems: A survey of surveys. IEEE Design & Test 34, 7–17.

[16] Giraldo, J., Urbina, D., Cardenas, A., Valente, J., Faisal, M., Ruths, J., Tippenhauer, N.O., Sandberg, H., Candell, R., 2018. A survey of physics-based attack detection in cyber-physical systems. ACM Computing Surveys (CSUR) 51, 76.

[17] Goh, J., Adepu, S., Junejo, K.N., Mathur, A., 2016. A dataset to support research in the design of secure water treatment systems, in: Int'l Conference on Critical Information Infrastructures Security, Springer. pp. 88–99.

[18] Goh, J., Adepu, S., Tan, M., Lee, Z.S., 2017. Anomaly detection in cyber physical systems using recurrent neural networks, in: 18th Int'l Symp. High Assurance Systems Engineering (HASE), IEEE. pp. 140–145.

[19] Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y., 2016. Deep learning. volume 1. MIT press Cambridge.

[20] Herzberg, A., Kfir, Y., 2019. The chatty-sensor: a provably-covert channel in cyber physical systems, in: Proc. 35th Annual Computer Security Applications Conference, pp. 638–649.

[21] Huang, L., Zhu, Q., 2020. A dynamic games approach to proactive defense strategies against advanced persistent threats in cyber-physical systems. Computers & Security 89, 101660.

[22] Humayed, A., Lin, J., Li, F., Luo, B., 2017. Cyber-physical systems security — a survey. IEEE Internet of Things Journal 4, 1802–1831.

[23] Inoue, J., Yamagata, Y., Chen, Y., Poskitt, C.M., Sun, J., 2017. Anomaly detection for a water treatment system using unsupervised machine learning, in: Int'l Conf. Data Mining Workshops, IEEE. pp. 1058–1065.

[24] Jia, Y., Wang, J., Poskitt, C.M., Chattopadhyay, S., Sun, J., Chen, Y., 2021. Adversarial attacks and mitigation for anomaly detectors of cyber-physical systems. International Journal of Critical Infrastructure Protection , 100452.

[25] Kim, J., Yun, J.H., Kim, H.C., 2019. Anomaly detection for industrial control systems using sequence-to-sequence neural networks, in: Computer Security. Springer, pp. 3–18.

[26] Kiss, I., Haller, P., Bereş, A., 2015. Denial of service attack detection in case of tennessee eastman challenge process. Procedia Technology 19, 835–841.

[27] Kravchik, M., Biggio, B., Shabtai, A., 2021. Poisoning attacks on cyber attack detectors for industrial control systems, in: Proceedings of the 36th Annual ACM Symposium on Applied Computing, pp. 116–125.

[28] Kravchik, M., Shabtai, A., 2018. Detecting cyber attacks in industrial control systems using convolutional neural networks, in: Proc. 2018 Workshop on Cyber-Physical Systems Security and PrivaCy, ACM. pp. 72–83.

[29] Kravchik, M., Shabtai, A., 2021. Efficient cyber attack detection in industrial control systems using lightweight neural networks and pca. IEEE Transactions on Dependable and Secure Computing .

[30] Krotofil, M., Cárdenas, A.A., 2013. Resilience of process control systems to cyber-physical attacks, in: Nordic Conference on Secure IT Systems, Springer. pp. 166–182.

[31] Krotofil, M., Larsen, J., 2015. Rocking the pocket book: Hacking chemical plants, in: DefCon Conference, DEFCON.

[32] Kushner, D., 2013. The real story of stuxnet. IEEE Spectrum 3, 48–53.

[33] Li, J., Yang, Y., Sun, J.S., Tomsovic, K., Qi, H., 2021. Conaml: Constrained adversarial machine learning for cyber-physical systems, in: Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, pp. 52–66.

[34] Lin, Q., Adepu, S., Verwer, S., Mathur, A., 2018. Tabor: a graphical model-based approach for anomaly detection in industrial control systems, in: Proc. 2018 on Asia Conf. Comp. Comm. Sec., ACM. pp. 525–536.

[35] Liu, J., Zhang, W., Ma, T., Tang, Z., Xie, Y., Gui, W., Niyoyita, J.P., 2020. Toward security monitoring of industrial cyber-physical systems via hierarchically distributed intrusion detection. Expert Systems With Applications 158, 113578.

[36] Maclaurin, D., Duvenaud, D., Adams, R., 2015. Gradient-based hyperparameter optimization through reversible learning, in: Int'l Conference on Machine Learning, pp. 2113–2122.

[37] Madani, P., Vlajic, N., 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. p. 1.

[38] Mitchell, R., Chen, I.R., 2014. A survey of intrusion detection techniques for cyber-physical systems. ACM Computing Surveys 46, 55.

[39] Muñoz-González, L., Biggio, B., Demontis, A., Paudice, A., Wongrassamee, V., Lupu, E.C., Roli, F., 2017. Towards poisoning of deep learning algorithms with back-gradient optimization, in: Proc. 10th ACM Workshop on Artificial Intelligence and Security, ACM. pp. 27–38.

[40] Nedeljkovic, D., Jakovljevic, Z., 2021. Cnn based method for the development of cyber-attacks detection algorithms in industrial control systems. Computers & Security , 102585.

[41] Noorizadeh, M., Shakerpour, M., Meskin, N., Unal, D., Khorasani, K., 2021. A cyber-security methodology for a cyber-physical industrial control system testbed. IEEE Access 9, 16239–16253.

[42] Pechenizkiy, M., Bakker, J., Žliobaitė, I., Ivannikov, A., Kärkkäinen, T., 2010. Online mass flow prediction in cfb boilers with explicit detection of sudden concept drift. ACM SIGKDD Explorations Newsletter 11, 109–116.

[43] Pires, W.R., de Paula Figueiredo, T.H., Wong, H.C., Loureiro, A.A.F., 2004. Malicious node detection in wireless sensor networks, in: 18th Int'l Parallel and Distributed Processing Symp., IEEE. p. 24.

[44] Raman, M.G., Somu, N., Mathur, A.P., 2019. Anomaly detection in critical infrastructure using probabilistic neural network, in: Int'l Conference on Applications and Techniques in Information Security, Springer. pp. 129–141.

[45] Ravikumar, G., Hyder, B., Govindarasu, M., 2020. Next-generation cps testbed-based grid exercise-synthetic grid, attack, and defense modeling, in: 2020 Resilience Week (RWS), IEEE. pp. 92–98.

[46] Rosenberg, I., Shabtai, A., Rokach, L., Elovici, Y., 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. pp. 490–510.

[47] Rubinstein, B.I., Nelson, B., Huang, L., Joseph, A.D., Lau, S.h., Rao, S., Taft, N., Tygar, J.D., 2009. Antidote: understanding and defending against poisoning of anomaly detectors, in: Proc. 9th ACM SIGCOMM conference on Internet measurement, pp. 1–14.

[48] Shafahi, A., Huang, W.R., Najibi, M., Suciu, O., Studer, C., Dumitras, T., Goldstein, T., 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks, in: Adv. Neural Inf. Proc. Sys., pp. 6103–6113.

[49] Shi, E., Perrig, A., 2004. Designing secure sensor networks. IEEE Wireless Communications 11, 38–43.

[50] Suciu, O., Marginean, R., Kaya, Y., Daume III, H., Dumitras, T., 2018. When does machine learning fail? generalized transferability for evasion and poisoning attacks, in: 27th USENIX Sec. Symp., pp. 1299–1316.

[51] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R., 2014. Intriguing properties of neural networks, in: International Conference on Learning Representations.

[52] Taormina, R., Galelli, S., 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, 04018065.

[53] Taormina, R., Galelli, S., Tippenhauer, N.O., Salomons, E., Ostfeld, A., Eliades, D.G., Aghashahi, M., Sundararajan, R., Pourahmadi, M., Banks, M.K., et al., 2018. Battle of the attack detection algorithms: Disclosing cyber attacks on water distribution networks. Journal of Water Resources Planning and Management 144, 04018048.

[54] Yin, S., Ding, S.X., Haghani, A., Hao, H., Zhang, P., 2012. A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark tennessee eastman process. Journal of process control 22, 1567–1581.

[55] Zizzo, G., Hankin, C., Maffeis, S., Jones, K., 2019. Adversarial machine learning beyond the image domain, in: 2019 56th ACM/IEEE Design Automation Conference (DAC), IEEE. pp. 1–4.

[56] Zizzo, G., Hankin, C., Maffeis, S., Jones, K., 2020. Adversarial attacks on time-series intrusion detection for industrial control systems, in: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE. pp. 899–910.

**Moshe Kravchik** received his M.Sc. (2007) in computer science from the Open University of Israel. He is currently pursuing a Ph.D. at Ben-Gurion University of the Negev. His research interests include the topics of software and systems security, trusted execution environments, anomaly detection, and the security of industrial control systems.

**Luca Demetrio** is a Postdoctoral Researcher in the Department of Electrical and Electronic Engineering at the University of Cagliari, Italy. He received his Ph.D. (2021), M.Sc. (2017), and B.Sc (2015) in Computer Science from the University of Genova. His scientific interests cover the overlapping between adversarial machine learning and computer security, with a strong focus on understanding the weaknesses of malware detectors.

**Battista Biggio** (Senior Member, IEEE) received the M.Sc. degree (Hons.) in electronic engineering and the Ph.D. degree in electronic engineering and computer science from the University of Cagliari, Italy, in 2006 and 2010, respectively. Since 2007, he has been with the Department of Electrical and Electronic Engineering, University of Cagliari, where he is currently an Assistant Professor. In 2011, he visited the University of Tuebingen, Germany, and worked on the security of machine learning to training data poisoning. His research interests include secure machine learning, multiple classifier systems, kernel methods, biometrics, and computer security. He is a member of IAPR. He also serves as a reviewer for several international conferences and journals.

**Asaf Shabtai** is a professor in the Department of Software and Information Systems Engineering at Ben-Gurion University of the Negev. His main areas of interest are computer and network security, machine learning, security of the IoT and smart mobile devices, and security of avionic and operational technology (OT) systems.