Evading ML Network Intrusion Detection Systems for Modbus TCP with Problem-Space Perturbations

Dimitri Galli[1,*,†], Giovanni Gambigliani Zoccoli[1,†], Daniele Bianchini[1], Dario Stabili[1] and Mirco Marchetti[1]

[1]University of Modena and Reggio Emilia, Department of Engineering "Enzo Ferrari", 41125 Modena, Italy

Abstract

Machine Learning (ML) plays a central role in Network Intrusion Detection Systems (NIDS), as it can be used to analyze complex traffic patterns and detect previously unseen attacks. With Industrial Control Systems (ICS) becoming increasingly connected and exposed to novel threats, ML-NIDS have been widely adopted to provide automated detection of cyberattacks targeting control processes and critical infrastructures. While ML-NIDS demonstrate good effectiveness and high performance, adversarial ML attacks based on input data manipulation have been shown to undermine their robustness. However, existing adversarial strategies against ML-NIDS in ICS primarily focus on modifying traffic attributes at the flow level, without considering the constraints imposed by the underlying network protocols. In this paper, we propose a packet-level adversarial attack to evade ML-NIDS for Modbus TCP. We modify raw packet captures using two manipulation methods, namely *jitter* and *padding*, which reshape the features of the resulting flows while preserving Modbus functionalities. We evaluate the impact of these perturbations on the CIC Modbus 2023 dataset, considering three different attacker scenarios and targeting three popular ML models (Decision Tree, Random Forest, Histogram Gradient Boosting). Results show that our attack strategies degrade ML-NIDS performance, with detection rates falling below 0.5 in 31 out of 45 cases.

Keywords

Adversarial Attacks, Machine Learning, Network Intrusion Detection Systems, Industrial Control Systems, Modbus TCP, Problem-Space Perturbations

1. Introduction

Industrial Control Systems (ICS) form the backbone of many critical infrastructures, monitoring and controlling physical processes in environments such as manufacturing plants and power grids [1, 2, 3]. These systems depend on low-latency and reliable communication between supervisory components and field devices to ensure continuous and safe operation [1, 4]. Among the industrial protocols in use, *Modbus TCP* remains one of the most widely used due to its simplicity, openness, and compatibility with legacy equipment [5, 6]. In this context, Human-Machine Interfaces (HMIs) and Supervisory Control and Data Acquisition (SCADA) systems act as Modbus *clients* that periodically poll Intelligent Electronic Devices (IEDs) and Programmable Logic Controllers (PLCs), which operate as Modbus *servers* [6, 7]. This polling behaviour produces highly regular request-response patterns that characterise Modbus TCP communications in many industrial networks [8]. However, Modbus TCP provides no authentication, encryption, or integrity protection, making it vulnerable to injection, replay, and spoofing attacks [6, 9].

To prevent cyber threats in industrial environments, network administrators increasingly rely on Network Intrusion Detection Systems (NIDS) based on Machine Learning (ML) methods. In typical deployments, ML-NIDS operate on raw packet traces or aggregated flow records and learn patterns that distinguish benign activity from malicious behavior [10, 11]. In this way, ML-NIDS analyze network traffic and detect anomalies that may correspond to unauthorized operations, raising alerts that are

Joint National Conference on Cybersecurity (ITASEC & SERICS 2026), February 09–13, 2026, Cagliari, IT

*Corresponding author.

†These authors contributed equally.

dimitri.galli@unimore.it (D. Galli); giovanni.gambiglianizoccoli@unimore.it (G. G. Zoccoli); daniele.bianchini@unimore.it (D. Bianchini); dario.stabili@unimore.it (D. Stabili); mirco.marchetti@unimore.it (M. Marchetti)

0009-0006-0280-2498 (D. Galli); 0009-0003-1700-5166 (G. G. Zoccoli); 0009-0008-5002-5456 (D. Bianchini); 0000-0001-6850-334X (D. Stabili); 0000-0002-7408-6906 (M. Marchetti)

forwarded to Security Operation Center (SOC) analysts. Compared to traditional signature-based approaches, ML-NIDS automate the detection process, generalize better to novel cyberattack variants, and offer greater flexibility in dynamic scenarios [12, 13].

In recent years, numerous papers have explored ML techniques for anomaly and intrusion detection in the context of ICS [14, 15, 16, 17, 18]. Most of these works propose ML classifiers that integrate complex learning mechanisms and obtain very high performance results in detecting different cyberattack variants, suggesting that ML can effectively discriminate between legitimate and malicious operations. However, these ML detectors are often evaluated only on clean data examples, assuming dataset distributions with well-separated benign and malicious classes. These evaluation settings may overstate the practical effectiveness of ML models for NIDS, as they do not fully represent the complexity and variability typical of real-world industrial environments.

Adversarial ML attacks [19, 20] against ML-NIDS have become a key topic in cybersecurity research [21, 22]. Indeed, an attacker can craft small and targeted perturbations to the input data in order to deceive the classifier and evade detection. However, adversarial robustness has received limited attention and remains largely underexplored in the literature on Modbus TCP. Only a small body of work investigates adversarial ML attacks against ML-NIDS for ICS, with most approaches operating on aggregated traffic attributes rather than on the original packet captures [23, 24, 25]. To evade ML-NIDS, attackers manipulate feature representations derived from network traffic without ensuring that the resulting modifications correspond to feasible packet sequences or valid protocol exchanges. In other words, attacks based on feature perturbation may overlook Modbus protocol constraints, such as admissible field values, request-response pairing, and function-code semantics.

In this paper, we address this gap by introducing a novel problem-space adversarial attack against ML-NIDS that analyze Modbus TCP traffic. Instead of manipulating derived feature vectors, we perturb the original Modbus TCP packets while enforcing operational constraints. Our approach relies on two practical perturbation mechanisms. The first, called *jitter*, delays communication by modifying packet timestamps and reshaping inter-arrival times. The second, named *padding*, injects junk bytes into the payload of Modbus frames without altering function codes or register values. After applying these perturbations under protocol-aware constraints, we export and generate the perturbed flow records, which preserve the malicious functionality and remain syntactically valid.

We evaluate our adversarial attacks on the CIC Modbus 2023 dataset [26], which includes benign and malicious Modbus TCP packets. The dataset simulates different threat conditions through three attacker configurations: (i) an attacker compromising a *device external* to the industrial network and transmitting malicious packets, (ii) an attacker controlling an *HMI* within the industrial network and sending malicious requests, and (iii) an attacker controlling an *IED* within the industrial network and sending malicious responses. Our adversarial perturbations target three popular supervised ML classifiers, namely Decision Tree (DT), Random Forest (RF), and Histogram Gradient Boosting (HGB). For each configuration, we train these models on labeled flow records to classify benign and malicious records. At inference time, we manipulate the PCAP traces, generate the perturbed flows, and measure the resulting degradation in detection rate. The results show that ML-NIDS with near-perfect accuracy on clean data can experience a dramatic performance drop when exposed to perturbed traffic, highlighting how easily adaptive attackers can evade detection.

This paper is organized as follows. First, Section 2 introduces background on ML-NIDS for Modbus TCP and compares our work with related research on adversarial ML attacks against ML-NIDS. Section 3 defines the target system and the threat model, formalizing attacker's goals, capabilities, knowledge, and strategy. Section 4 presents the proposed adversarial attack methodology. Section 5 describes the case study, including the dataset, the ML detectors, and the attack implementation. Section 6 details the evaluation setup and reports the results. Finally, Section 7 concludes the paper and outlines directions for future research.

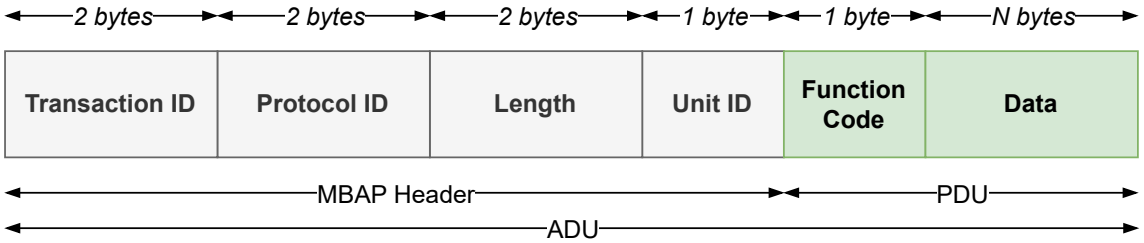

Figure 1: Fields of a Modbus TCP message. The fields colored gray belong to the header, while the fields colored green belong to the payload.

2. Background and Related Work

This section provides the technical background and reviews related work. We begin by describing common communication patterns in ICS and some Modbus TCP protocol specifications (Section 2.1). Next, we present an overview of ML-NIDS in ICS environments (Section 2.2). We then examine adversarial ML attacks against ML-NIDS and their limitations when applied to ICS settings (Section 2.3).

2.1. ICS and Modbus TCP

In practical deployments, ICS employ a two-layer communication model: (i) a *control* or *supervisory* layer, where HMIs and SCADA systems operate, and (ii) a *field* or *production* layer, where IEDs and PLCs interface with the physical processes [1, 27]. The control layer implements monitoring logic and issues control decisions. The field layer executes these decisions by acquiring sensor data and actuating physical components such as motors and valves. Modbus TCP operates on top of the TCP/IP stack and provides a lightweight request-response protocol for regulating the exchange of measurements, commands, and control messages between the devices in these layers. Modbus TCP extends the original serial Modbus specification while also integrating naturally with high-speed Ethernet networks. Modbus TCP has gained widespread adoption among operators and vendors due to its full compatibility with legacy devices.

Figure 1 illustrates the different fields of a typical Modbus TCP message, which consists of a *header* followed by a variable-length *payload* [28]. The Modbus Application Protocol (MBAP) header comprises various identifiers and the length field. These fields enable clients to correlate requests and responses, as well as to query specific field devices, including those behind gateways. The payload, or Protocol Data Unit (PDU), contains the field of the function code and operation-specific data (e.g., target addresses, operation ranges, and transmitted values) and supports both single-register and multi-register read/write operations.

The communication pattern induced by Modbus TCP is largely cyclic [6, 8]. HMIs, SCADA systems, and workstations typically act as Modbus *clients* (or *masters*) that poll field devices (e.g., IEDs and PLCs) on fixed time schedules, issuing read commands for the same registers repeatedly and sending write requests only occasionally to update setpoints or control outputs. Instead, IEDs and PLCs execute as Modbus *servers* (or *slaves*) by exposing control points and process variables [6, 7]. This yields highly regular request-response sequences and strongly periodic traffic profiles. However, Modbus TCP lacks any mechanism for authentication, confidentiality, or message integrity [29]. As a consequence, any host with access to the industrial network can forge or replay malicious requests or responses [5, 6]. Prior works have shown that attackers can exploit this lack of security to manipulate register values, issue arbitrary commands, or desynchronize operator views from the actual process state through injection and replay attacks [5, 9]. These risks motivate the deployment of intrusion detection solutions capable of recognizing anomalous behavior and reporting suspicious operations [6, 14].

2.2. ML-NIDS

NIDS address the limited security of industrial protocols by monitoring network traffic and raising alerts when observed events match human-defined rules (*misuse-based* approach) or when tracked patterns deviate from the expected "normal" profile (*anomaly-based* approach). While NIDS based on predefined signatures are very precise, they have proven insufficient in detecting new cyber attacks or modified variants of known cyber threats [10]. Conversely, anomaly-based NIDS tend to generate more false positives, but they are more likely to detect novel or zero-day attacks [30]. As network technologies evolve and traffic volumes increase, *data-driven* solutions, such as ML, have been proposed to automate the detection process, yielding improvements that in some cases outperform traditional detection systems [31].

In the context of ICS, ML-NIDS analyze network traffic to identify anomalies and protect industrial devices [32]. An ML-NIDS is a detection system that employs a trained ML model within the following operational pipeline [13, 33]. First, raw network packets are captured and collected by traffic sensors located in the internal network (e.g., strategically placed taps or mirrored switch ports). Second, a preprocessing component transforms the input data into a format accepted by the ML model. For instance, PCAP traces can be aggregated into NetFlows [34], i.e., a common data format in which each record reports the communication between two endpoints over a given time window through metrics and statistics (e.g., *duration*, *byte count*, *packet size*). Finally, the ML model (e.g., Decision Tree, Random Forest, Support Vector Machine) previously trained detects malicious activity and raises alerts by distinguishing benign from anomalous traffic.

Several recent studies propose ML-NIDS for Modbus TCP [17, 18, 35, 36, 37]. These works explore both ML and DL models, including supervised binary classifiers, as well as unsupervised anomaly detection techniques. In these papers, detectors are trained on public ICS or Modbus datasets and evaluated exclusively on clean data from the test split. They consistently report very high performance results, suggesting strong discriminative capabilities and generalization to unseen attacks. However, these works do not consider adversarial scenarios in which attackers intentionally shape Modbus traffic to deceive the ML classifier and evade detection. Hence, evaluation of ML-NIDS robustness to adversarial perturbations represents a critical gap in current ICS security research.

2.3. Adversarial ML Attacks

A large body of recent literature has shown that ML models are intrinsically vulnerable to *adversarial ML attacks* [20, 38]. An attacker can manipulate input data with small targeted perturbations to deceive an ML model and obtain a desired outcome [39]. Such adversarial manipulations may be crafted as *poisoning* attacks [40] (i.e., by injecting modified samples into the training data to corrupt the learned decision boundary), or as *evasion* attacks [19] (i.e., by perturbing malicious data to induce misclassification at inference time). As our objective is to evaluate ML-NIDS robustness at test time, we consider only *evasion* attacks throughout this work.

The vulnerability of ML models to adversarial perturbations is especially concerning for security applications, as unauthorized operations or undetected intrusions can lead to severe consequences [41, 42]. In the context of network intrusion detection, attackers can perturb their malicious communications to deceive the ML model, evade the NIDS, and maintain control over the compromised devices in the network [43, 44, 45]. However, most existing research on adversarial ML targeting ML-NIDS (e.g., [46, 47, 48, 49]) applies perturbations directly to the feature representation ingested by the ML model, i.e., in the so-called *feature space*, often leveraging optimization-based techniques such as *FGSM* [50] or *PGD* [51]. For example, perturbations can be generated by adding fixed amounts to the attributes of flows exported from network packets [25] or to the embeddings learned by the model [49]. While these approaches are very effective under the assumption that the envisioned attacker knows the exported features or the model parameters, it is well known that such manipulations may not be physically feasible in the real world or may not translate into valid modifications in the traffic space [22]. In a practical deployment, the flow exporter is typically placed with the ML model in the detection

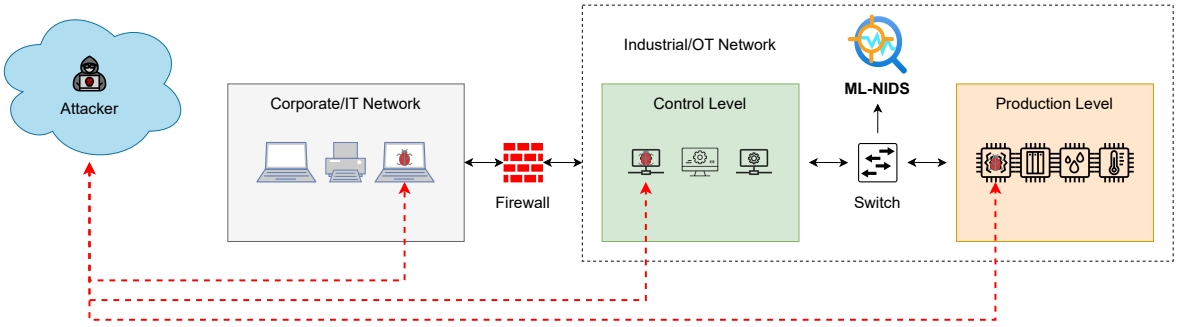

Figure 2: Threat model. The attacker controls only one machine at a time: (i) a device external to the OT network, (ii) an internal HMI at the control level, or (iii) an internal IED at the production level.

system (which is highly protected and not visible from outside the network [52]), hence a real-world attacker may not be aware of the full set of attributes used by the ML model [53]. In addition, as feature-space manipulations focus on modifying numerical values, they typically do not consider any protocol constraints for building valid packets. Hence, the resulting adversarial samples may correspond to messages that fail to preserve the original malicious functionality or include inconsistencies in header fields.

Rather than manipulating numerical feature vectors, recent studies (e.g., [54, 55, 56, 57]) apply perturbations in the *problem space*, i.e., by altering malicious packets generated by attacker-controlled hosts. In this way, attackers ensure that the perturbed traffic remains semantically consistent and syntactically valid under the underlying protocol. However, most works proposing problem-space perturbations against ML-NIDS focus on IT/IoT networks and do not explicitly address industrial/OT scenarios. Consequently, they do not enforce Modbus-specific requirements such as consistent MBAP headers, permitted function codes, or valid register combinations.

In this paper, we propose a packet-level adversarial attack against ML-NIDS for Modbus TCP through perturbations in the problem space. In particular, we manipulate the original PCAP traces using two perturbation strategies (i.e., jitter and padding) that attackers could realistically implement on their compromised devices. To craft valid packets that preserve the malicious intent, we respect protocol-aware constraints, like enforcing a minimum time separation between consecutive messages and ensuring packet length does not exceed the maximum allowed size. In this way, we generate protocol-valid adversarial traffic that significantly challenges the robustness of ML-NIDS.

3. Threat Model

We now define the threat model for this work. We first describe the target system (Section 3.1), then characterize the attacker's goal, knowledge, capabilities, and strategies (Section 3.2).

3.1. Target System

We consider a typical industrial network scenario for intrusion detection, as depicted in Figure 2. The internal network spans two hierarchical layers: the *control* level, where HMIs/SCADA systems monitor physical processes, and the *production* level, where IEDs/PLCs execute field operations. The network is monitored by an ML-NIDS, which represents the attacker's target system. This ML-NIDS consists of a *flow exporter*, which generates flow records from PCAP traces, and a *supervised ML classifier*, which detects malicious traffic. The flow exporter collects and extracts flow records from Modbus TCP packets forwarded through the mirror port of the industrial switch. The ML classifier constitutes the core detection component and is trained to distinguish benign from malicious Modbus flows.

3.2. Envisioned Attacker

We consider an envisioned adversary that, through prior compromise (e.g., an exploit), has obtained control of one host within the network. The attacker controls one machine at a time: (i) a device external to the OT network, (ii) an internal HMI at the control level, or (iii) an internal IED at the production level. The adversary's **goal** is to evade the ML-NIDS, i.e., to have their malicious communications misclassified as benign. Indeed, the attacker needs to evade detection in order to remain undetected and pursue their primary objective (e.g., sabotage).

The adversary has limited **knowledge** of the target ML-NIDS. The attacker's knowledge approximates a realistic *gray-box* scenario [57]. In particular, the adversary knows that a NIDS powered by an ML classifier, which has been trained using both legitimate and malicious flows, monitors the network. However, the attacker does not know either the training set or the model configuration (e.g., exact feature set, internal parameters, output probabilities).

The adversary has the **capability** to fully control their infected machines. However, the attacker cannot manipulate their malicious communications beyond the controlled node. Therefore, the adversary can craft perturbations only at the packet level. Moreover, the attacker cannot observe the output of the ML-NIDS, as it is highly protected and operates in a separate network segment.

The adversary has two **strategies** for evading the ML-NIDS: (i) add delays during packet transmission (i.e., *jitter*) or (ii) inject junk bytes into the packet payloads (i.e., *padding*). On the one hand, the jitter strategy introduces bounded communication delays in Modubs requests/responses by manipulating packet timestamps, which reshape inter-arrival times while preserving message order. On the other hand, the padding strategy perturbs Modbus messages by appending bytes to their payload and updating the header length field so that size-related statistics remain consistent with the payload content. We remark that both the attacker's strategies occur within the problem space under realistic protocol-aware constraints.

4. Adversarial Attack Methodology

This section presents our problem-space adversarial attack methodology. We first introduce notation and scope (Section 4.1), then describe our jitter (Section 4.2) and padding (Section 4.3) perturbation strategies.

4.1. Notation and Scope

A Modbus communication between two devices forms a *time-ordered sequence of packets* $\mathcal{P} = (t_i, x_i)_{i=1}^{N}$, where t_i is the timestamp of packet i and x_i is its content. Each message corresponds to a *request* or a *response* exchanged within a Modbus transmission between a client/master and a server/slave. *Benign* packets are associated with legitimate operations and are denoted with $\mathcal{B}$, while *malicious* packets are generated by infected machines in our attack scenarios and are denoted with $\mathcal{M}$. As described in Section 2, each packet carries a *header* H_i, which encodes protocol metadata and length fields, and a *payload* P_i, which includes the function code and operation-specific data.

Figure 3 depicts our adversarial attack methodology. We illustrate a Modbus communication of seven packets, where B_{t1}, B_{t3}, B_{t4}, B_{t5}, and B_{t7} are benign packets represented with green icons, whereas M_{t2} and M_{t6} are malicious packets represented with red icons. As outlined in Section 3, we assume a constrained attacker seeking to evade detection at inference time. In this context, the attacker leverages three perturbation strategies:

- **Jitter:** We add a bounded delay to malicious packets in the Modbus communication. From a practical viewpoint, we add a random or static value to the timestamp t_i of all malicious packets. This manipulation modifies the temporal spacing between packets to mimic natural network variability. In Figure 3, malicious packets $M_{t'2}$ and $M_{t'6}$ are shifted with a delay that does not exceed the timestamp of the immediately following packet to be transmitted over the wire.

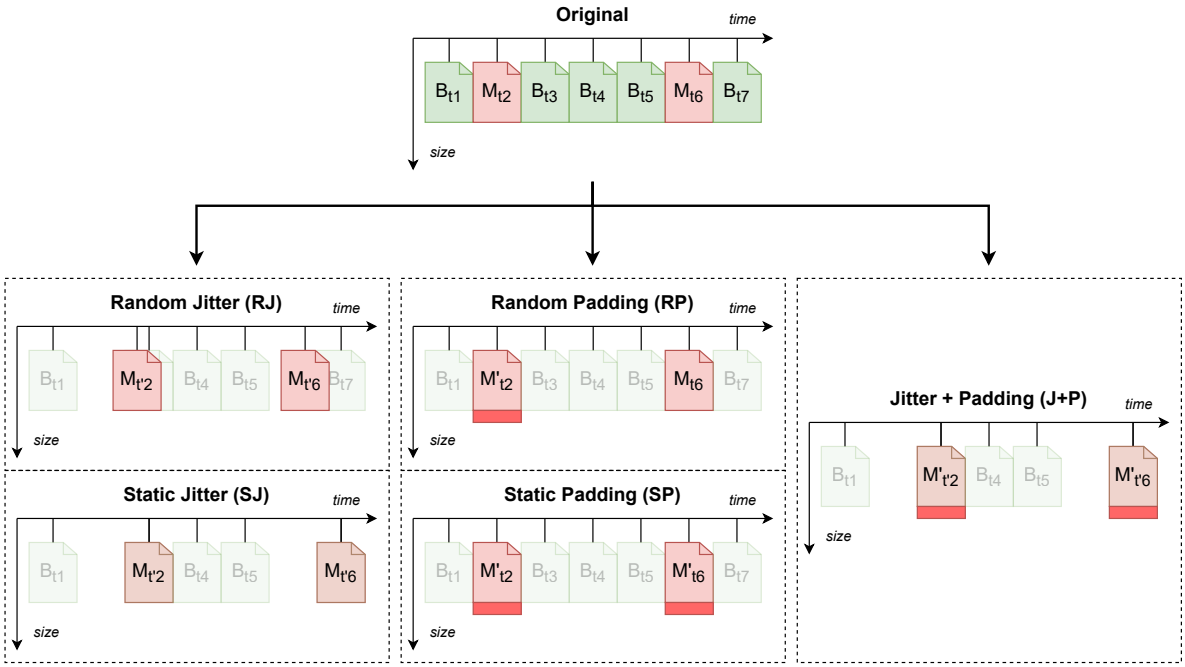

Figure 3: Adversarial attack methodology. From an original sequence of benign and malicious packets, we apply *jitter* (time shifts) and *padding* (size increases) perturbations—either randomly or statically—to generate protocol-valid adversarial messages that alter the extracted flow-level features.

- **Padding:** We append junk bytes to the packet payload. More specifically, we inject null bytes into the content x_i of malicious packets without exceeding the maximum message size. This changes the exported feature values while keeping intact protocol semantics. In Figure 3, the malicious packet M'_{t2} is always padded, while the malicious packet M'_{t6} is padded only in the static case.
- **Jitter + Padding:** We apply jitter and padding perturbations simultaneously to the malicious messages, altering both size and temporal features of the extracted flows. In Figure 3, packets $M'_{t'2}$ and $M'_{t'6}$ are both delayed and padded with static manipulations.

We design and execute these adversarial manipulations at the packet level, i.e., we perturb raw Modbus TCP messages before exporting the flow records and submitting them to the ML model. Operating directly on PCAP traces in the traffic space requires preserving the communication logics and respecting the protocol semantics. Hence, while performing perturbations, we enforce the following constraints: (i) the original order of packets must not change, (ii) requests must always precede corresponding responses, (iii) the length of packets must not exceed the maximum size allowed by the protocol, and (iv) headers must remain consistent and well-formed. In this way, we ensure that our perturbations preserve both the functional impact and malicious intent of the modified packets, still transferring to the feature space, where they influence the flow-level attributes ingested by the ML-NIDS.

4.2. Jitter

The jitter perturbation alters the timestamps t_i of malicious packets $\mathcal{M}$. We introduce a bounded delay that modifies the time-related features (e.g., *flow duration*) of the client-server communication without changing the packet content x_i. With this adversarial manipulation, we simulate an attacker attempting to mimic natural network conditions (e.g., congestion, queuing) in order to evade the ML-NIDS. We define the jitter perturbation as follows:

$$t'_i = t_i + \Delta t \tag{1}$$

where t_i is the original timestamp of packet i, Δt is the introduced delay, and t'_i is the resulting perturbed timestamp.

Industrial networks and OT scenarios exhibit both random delays and stable latency fluctuations. Hence, we develop two different jitter perturbation variants:

- **Random Jitter (RJ):** We delay the actual malicious packet i with a random offset sampled from a uniform distribution bounded by the time interval between the timestamp t_i of packet i and the timestamp t_{i+1} of the following packet $i+1$ to transmit. In this way, we model an adaptive attacker that introduces unpredictable timing variations to blend malicious traffic with natural network fluctuations.
- **Static Jitter (SJ):** We delay the actual malicious packet i with a static offset computed as the time difference between the timestamp t_i of packet i and the timestamp t_{i+1} of the next packet $i+1$. In this way, we model a systematic attacker that applies a deterministic and reproducible timing shift, pushing each malicious message as close as possible to the next transmission event.

To avoid unrealistic temporal collisions, we enforce a minimum time separation between transmission of consecutive packets. In this way, jittered packets do not collapse onto identical timestamps, thereby maintaining the original request-response pattern typical of a Modbus TCP communication.

4.3. Padding

In padding perturbation, we append junk data to the payload P_i of malicious messages $\mathcal{M}$ and update the corresponding length fields in the header H_i. In this way, we change various measures and statistics (e.g., *exchanged bytes*) of the communication that are reflected in the features of the flow records analyzed by the ML model. With this adversarial attack, we simulate an attacker who is aware of extant research and knows that tiny perturbations in size-related features can easily evade the ML-NIDS [58]. More formally, we define the padding perturbation as:

$$P'_i = P_i \| \Delta P \tag{2}$$

where P_i is the original packet payload, ΔP is the junk data used for injection, and P'_i is the padded payload.

An attacker may choose to manipulate packets either sporadically to remain less predictable or systematically to maximize the shift in size-related statistics. Therefore, we propose two diverse padding perturbation variants:

- **Random Padding (RP):** We pad with junk bytes ΔP a random half of the total number of malicious messages. In this way, we model a constrained attacker that avoids a fully stationary perturbation strategy.
- **Static Padding (SP):** We pad with junk data ΔP all malicious packets. In this way, we model a more aggressive attacker that systematically perturbs their communications to maximize evasion potential.

To comply with communication constraints and Modbus specifications, we pad only the fields that allow variable-length data (i.e., PDUs of read/write operations). After padding, we also ensure that each packet does not exceed the maximum length allowed by the Modbus TCP protocol, which is 260 bytes (7 bytes for the MBAP header and 253 bytes for the PDU). Since the header must be consistent with the new content of the packet, we recompute the MBAP length field in order to reflect the updated payload size. When the payload includes internal length counters (e.g., byte-count field in responses to function code `0x03` or in requests/responses for `0x10`), we also update these counters to maintain well-formatted frames.

5. Case Study

We now describe the case study where we implement our adversarial attack methodology against ML-NIDS for Modbus TCP. We present in detail the dataset (Section 5.1), the ML detectors (Section 5.2), and the implementation of our perturbation strategies (Section 5.3).

5.1. Dataset

We adopt the **CIC Modbus dataset** [26] for our experimental case study. This dataset includes both *benign* and *malicious* Modbus TCP packets collected from a simulated Docker-based testbed. As depicted in Figure 4, each container emulates an industrial device (e.g., HMI or IED) that transmits or receives packets on the Modbus network. The benign data consists of PCAP traces representing legitimate Modbus communications within a substation environment. The malicious portion contains PCAP traffic obtained while executing different cyberattacks aligned with the MITRE ICS ATT&CK framework (e.g., false data injection, parameter-length modification, query flooding, reconnaissance). Malicious data has been collected under three attacker configurations that depend on the machine controlled by the attacker: (i) an *external device* (with IP 185.175.0.7) outside the industrial network perimeter, (ii) an *internal HMI* at the control level (with IP 185.175.0.3), or (iii) an *internal IED* at the production level (with IP 185.175.0.5). For each attack scenario, the attacker controls the corresponding device and employs it to generate malicious packets towards other machines within the network. This dataset is gaining increasing popularity in the literature [17, 18] because it provides labeled PCAP traces, making it suitable for building supervised ML models capable of detecting malicious Modbus TCP traffic.

At the beginning of preprocessing, we verify the integrity and ordering of packets using *pcapfix*[1] and *reordercap*[2]. The first tool corrects corrupted PCAP traces, while the second one fixes timestamp inconsistencies and reorders packets when necessary. Once the network traffic is validated, we employ the fixed version of *CICFlowMeter* [59, 60] to transform each packet capture into a collection of flow records with more than 80 attributes (e.g., flow duration, byte count, packet size). According to the ground-truth annotations provided in the dataset documentation, we label the flows as benign or malicious. The dataset also includes non-Modbus traffic, so we filter the flow records to retain only Modbus TCP samples transmitted over port 502. To prevent memorization of device identity and reduce spurious correlations that could bias the ML model [61], we remove IP addresses, port numbers, and other metadata (e.g., flow identifiers, protocol tags, timestamps). Finally, we build three scenario-specific data collections by combining benign flows with the malicious flows from each attacker configuration. When assembling each dataset, we employ a benign:malicious ratio of 20:1 to approximate a realistic ICS class distribution. Each collection is then partitioned into training and test sets with an 80:20 stratified split. Each training set contains 800 000 benign flows and 40 000 malicious flows, while each test set includes 200 000 benign records and 10 000 malicious records.

5.2. ML Detectors

To evaluate the impact of our adversarial perturbations against ML-NIDS, we consider three ML algorithms: **Decision Tree (DT)**, **Random Forest (RF)**, and **Histogram Gradient Boosting (HGB)**. We select these ML models because they are widely used in intrusion detection [33, 62], and previous work has shown that they provide good performance results on Modbus traffic classification [18].

DT partitions the feature space using axis-aligned thresholds to maximize class separation, yielding an interpretable model that captures simple decision rules. RF and HGB are ensemble tree-based models that aggregate multiple learners to improve generalization on high-dimensional tabular data. RF combines multiple trees fitted on bootstrap samples and random feature subsets, limiting overfitting and reducing variance. HGB trains the ensemble of trees in a stage-wise strategy on binned feature values, capturing non-linear relationships and optimizing the loss.

[1] https://github.com/Rup0rt/pcapfix
[2] https://www.wireshark.org/docs/man-pages/reordercap.html

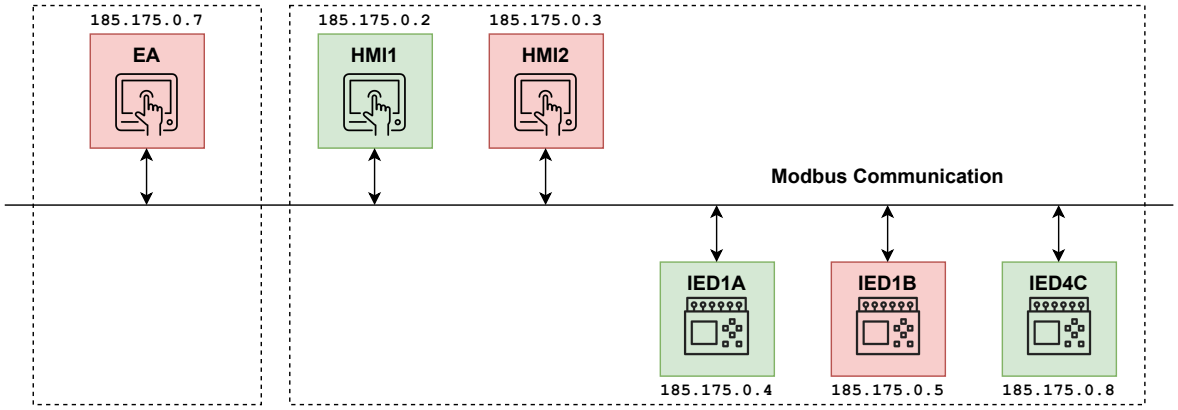

Figure 4: CIC Modbus 2023 dataset testbed [26]. Packets are collected under three attack scenarios, each of which depends on the device (colored red) controlled by the attacker.

We implement a binary classifier for each attack configuration considered in our dataset, i.e., an attacker controlling an external device, an internal HMI, or an internal IED. For each threat scenario, we treat all attack types as a single malicious class. In particular, having built 3 data collections (external host, internal HMI, and internal IED) and considered 3 different ML models (DT, RF, and HGB), we develop a total of $3 \times 3 = 9$ scenario-specific ML-NIDS. Each ML instance classifies the flow examples exported from the PCAP traces as either *benign* or *malicious*. For the hyperparameters of our chosen ML models, we use the *Gini impurity* criterion with no maximum depth for DT, 100 *trees* for RF, and a *log_loss* objective with learning rate 0.1 and 100 boosting iterations for HGB.

5.3. Attack Implementation

We implement our attack methodology in *Python*, leveraging *Scapy* for packet manipulation, *pandas* for preprocessing flow records, and *scikit-learn* for training and evaluating ML models. We release the code and results in our public repository: https://github.com/dimgalli/evading-ml-nids.git.

We begin by transforming each packet capture into a collection of flow records. As described in Section 5.1, we build 3 different datasets, each combining legitimate with malicious flows in a 20:1 ratio and reflecting a distinct attacker configuration. We split each collection into training and test sets using an 80:20 split. As detailed in Section 5.2, we train each binary classifier on the clean training set corresponding to a specific attacker configuration and evaluate the performance on the respective unperturbed test set.

After evaluating classifier performance on clean data, we craft perturbed data. For each attack configuration, we consider its malicious PCAP trace and filter Modbus TCP traffic generated by the corresponding infected host (IP address 185.175.0.7 for the external device, 185.175.0.3 for the internal HMI, and 185.175.0.5 for the internal IED) with destination/source port 502. Legitimate packets are copied unchanged to preserve the ordering and structure of the original PCAP trace. We then generate five adversarial trace variants by applying the five manipulation methods outlined in Section 4: *random jitter*, *static jitter*, *random padding*, *static padding*, and *jitter + padding*. For the jitter manipulation described in Section 4.2, we compute the time differences between consecutive packets and modify timestamps by adding either a random value sampled within the corresponding inter-packet interval (random variant) or the full interval as a fixed offset (static variant). In the padding perturbation presented in Section 4.3, we pad the payload of malicious Modbus messages by appending junk bytes (set to `0x00`) to either a randomly selected half of the malicious packets (random variant) or all malicious packets (static variant). After each perturbation, we apply syntactic checks and update relevant fields in the header to ensure compliance with the Modbus TCP protocol. Malformed messages that include inconsistencies are directly discarded. Having obtained the adversarial packets, we apply the same pipeline for exporting and preprocessing the adversarial flow records. We then perform the same

Table 1
Baseline results. We evaluate the ML-NIDS on clean flows and measure their *F1*, *Precision*, and *Recall* scores.

ML Classifier	External DEV			Internal HMI			Internal IED		
	F1	*Precision*	*Recall*	*F1*	*Precision*	*Recall*	*F1*	*Precision*	*Recall*
DT	1.000	0.999	0.999	0.842	0.840	0.844	0.906	0.907	0.905
RF	1.000	1.000	1.000	0.855	0.881	0.830	0.934	0.947	0.921
HGB	1.000	1.000	1.000	0.857	0.949	0.781	0.937	0.928	0.948

train-test split to ensure that adversarial robustness is evaluated only on perturbed flows derived from the test set. Any perturbed example that would fall into the training set is excluded from the experiments to avoid testing the model on flow records seen during training. In this way, we enable a fair comparison between detector performance on clean test flows and on their perturbed counterparts.

6. Evaluation and Results

This section describes the evaluation setup and discusses the findings of our case study. Based on previous work [63], we first evaluate the ML-NIDS on clean data to establish baseline performance (Section 6.1) and then on perturbed flows to measure adversarial resilience (Section 6.2).

To establish baseline performance, we evaluate the classifiers on clean test sets with benign and malicious flows derived from the original packets. Our goal is to verify that ML-NIDS achieve high detection performance on unperturbed network traces. To quantify adversarial resilience against our perturbations, we evaluate the detectors exclusively on test sets with adversarial flows obtained from the perturbed malicious PCAP. In this case, our objective is to compare the resilience of the ML-NIDS across the three attacker configurations and measure how much our perturbations degrade detection.

To evaluate the baseline performance of ML-NIDS on clean data, we rely on evaluation measures commonly used in network intrusion detection, i.e., *F1-score*, *Precision*, and *Recall*. These three metrics are defined as follows:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN}$$

where *TP*, *FP*, and *FN* denote *true positives*, *false positives*, and *false negatives*, respectively. To assess the effectiveness of our attacks, we measure the robustness of the target ML-NIDS using *Recall* (also denoted as *Detection Rate*, or *DR*), which indicates how many perturbed flows are correctly classified as malicious. In the rest of this work, we treat the malicious class as the positive class, so that a positive detection always corresponds to a malicious sample.

6.1. Baseline Results

Across each of the three attacker configurations, we evaluate the baseline performance of DT, RF, and HGB classifiers on the clean test sets, which contain both benign and malicious flows. This assessment ensures that our ML-NIDS exhibit high detection before perturbing traffic and submitting it to the trained detectors.

We report the baseline results in Table 1. Each classifier is associated with a row, while each column corresponds to one of the three attacker scenarios and includes the ML-NIDS performance in terms of *F1-score*, *Precision*, and *Recall*. The three classifiers achieve values very close to 1 across all metrics when the attacker comprises an external device. Even when the attacker controls a machine within the OT network (e.g., HMI or IED), all detectors exhibit good performance, with *F1-scores* above 0.8.

In general, we observe that all ML-NIDS provide very good results on unperturbed network traffic. Our findings align with previous work [18] and enable us to evaluate the impact of the proposed perturbation strategies.

6.2. Adversarial Results

After assessing the performance of DT, RF, and HGB classifiers on clean data, we focus on our main objective, i.e., evaluating the impact of our adversarial attack against ML-NIDS by measuring their resilience to adversarial perturbations. We recall that we do not apply manipulations directly to the numerical values of feature vectors. As pointed out in the threat model presented in Section 3, it is unfeasible for an attacker to manipulate the exported flow records, since they can only operate on their own controlled hosts. As outlined in Section 4, we modify raw malicious packets, transform the manipulated traces into flows, and use only the perturbed records belonging to the original test set for the evaluation.

We present our findings in the following sections. Each section corresponds to a specific perturbation strategy proposed in this work. We report the results in terms of *DR*, since it effectively captures how much the proposed attacks evade the classifiers. To illustrate the impact, *DR* scores are displayed in cells with varying shades of red, with darker shades indicating a greater decrease in *DR* and reflecting a more effective evasion strategy.

6.2.1. Jitter Results

We evaluate the ML-NIDS on adversarial flows generated using the *jitter* perturbation (Section 4.2). We consider two variants: (i) *random jitter*, where the delay is randomly sampled within the inter-packet interval between consecutive packets, and (ii) *static jitter*, where each packet timestamp is shifted close to the timestamp of the immediately following packet.

We present the results in Table 2. We immediately note that the detectors exhibit markedly different performance across the three attack scenarios. When the adversary controls an external host, jitter perturbations have little to no impact on the target detectors, with all *DR* scores above 0.99. This low effectiveness can be attributed to the temporal variability that is typical of packets exchanged across different network segments, where queuing and routing already introduce timing fluctuations. Conversely, we observe a significant decline in *DR* when the adversary controls a device located inside the OT network (i.e., the HMI or the IED). The worst performance occurs when the attacker controls the IED and applies the static jitter perturbation to malicious packets, with *DR* values dropping below 0.05.

Overall, these results highlight the susceptibility of ML-NIDS to timing manipulations performed by compromised devices within the monitored network, whereas the same perturbation strategy is largely ineffective when carried out from an external host.

6.2.2. Padding Results

We now evaluate the ML-NIDS on adversarial samples generated by *padding* the payload of malicious traces (Section 4.3). Again, we consider two variants: (i) *random padding*, where we pad half of the total number of malicious packets, and (ii) *static padding*, where we pad all malicious packets sent from the attacker-controlled host.

We report the results in Table 3. We observe a general degradation in *DR* across all detectors and attack scenarios. The static padding variant leads to the strongest degradation in *DR*, especially when the adversary controls the IED. In this case, all detectors exhibit very poor detection performance, with *DR* values below 0.2. Random padding yields slightly higher *DR* values than the static strategy, yet it still causes a substantial drop compared to the baseline. For example, HGB, which achieves a baseline *DR* of 1.000, 0.781, and 0.948 across the three configurations, experiences a marked reduction after padding, with *DR* values of 0.520, 0.362, and 0.357 in the same scenarios. However, when the attacker controls the HMI, padding is less effective. Under the random variant, DT, RF, and HGB achieve a *DR* of 0.645, 0.702, and 0.362, respectively, whereas under the static variant, the three classifiers yield *DR* values of 0.690, 0.786, and 0.418.

Overall, padding perturbations degrade the performance of the ML detectors, with the static strategy being the most effective, particularly when the attacker controls the IED.

Table 2
Jitter results. We evaluate the ML-NIDS on flows manipulated with jitter perturbation and measure their *DR* scores.

ML Classifier	Random Jitter (RJ)			Static Jitter (SJ)		
	External DEV	*Internal HMI*	*Internal IED*	*External DEV*	*Internal HMI*	*Internal IED*
DT	0.999	0.292	0.256	0.999	0.271	0.020
RF	1.000	0.247	0.163	0.999	0.216	0.012
HGB	0.998	0.208	0.158	0.996	0.183	0.000

Table 3
Padding results. We evaluate the ML-NIDS on flows manipulated with padding perturbation and measure their *DR* scores.

ML Classifier	Random Padding (RP)			Static Padding (SP)		
	External DEV	*Internal HMI*	*Internal IED*	*External DEV*	*Internal HMI*	*Internal IED*
DT	0.372	0.645	0.409	0.270	0.690	0.108
RF	0.410	0.702	0.368	0.000	0.786	0.034
HGB	0.520	0.362	0.357	0.339	0.418	0.000

6.2.3. Jitter + Padding Results

After evaluating the performance on data manipulated using individual strategies, we test the ML-NIDS on flows modified by combining *jitter* and *padding* perturbations. We apply the *static* version of both strategies: for each malicious packet, we shift its timestamp close to the timestamp of the next packet and pad its payload up to the maximum size (i.e., 253 bytes).

We show the results in Table 4. Across the three attack scenarios, HGB proves to be the most vulnerable classifier to our combined perturbations, with *DR* scores dropping to 0.247, 0.196, and 0.348. When the adversary controls the external host, the three classifiers exhibit a significant performance decrease, with *DR* values below 0.25. Conversely, the jitter + padding perturbation against DT and RF is mostly ineffective when the attacker operates from the HMI. Indeed, DT and RF experience only a minor decrease in performance, with *DR* results of 0.779 and 0.823, which are very close to their baseline scores. This limited impact may be explained by the simultaneous increase in payload size and transmission duration, which together define perturbations that closely resemble legitimate traffic variability.

Overall, combining jitter and padding yields general performance degradation, although in scenarios with highly regular traffic patterns, the impact on some detectors remains limited.

7. Conclusion

The adoption of ML algorithms for NIDS in industrial environments has garnered significant attention. The majority of ML-NIDS observe the features of flow records exported from network packets and raise security alerts when they detect deviations from expected patterns. Despite the ability to automate the detection task, several studies have shown that ML-NIDS are susceptible to adversarial manipulation of network traffic, enabling attackers to evade these systems. However, most existing research focuses on adversarial attacks that directly perturb flow records, which are typically unavailable to attackers in real-world deployments.

In this paper, we introduced a packet-level adversarial attack against ML-NIDS for Modbus TCP. We proposed two perturbation methods (i.e., jitter and padding) that model realistic evasion attempts in the problem space while preserving Modbus protocol constraints. The effectiveness of our attacks is evaluated on the CIC Modbus 2023 dataset under three adversary configurations, simulating attackers that control either an external device or compromised machines inside the OT network. Our results reveal that both jitter and padding can easily evade detection, particularly when the attacker controls industrial devices located inside the monitored network.

Table 4
Jitter + padding results. We evaluate the ML-NIDS on flows manipulated with jitter and padding perturbations, and measure their *DR* scores.

ML Classifier	Jitter + Padding (J+P)		
	External DEV	*Internal HMI*	*Internal IED*
DT	0.235	0.779	0.619
RF	0.000	0.823	0.273
HGB	0.247	0.196	0.348

This study highlights the importance of evaluating ML-NIDS robustness under problem-space attacks. Future work should investigate how different perturbation magnitudes (e.g., delay ranges and padding sizes) affect detection performance and develop countermeasures to enhance robustness.

Acknowledgments

This work has been supported by the project C4SI funded by the Emilia-Romagna region within the PR-FESR 2021-2027 (CUP E67G22000630003).

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] K. Stouffer, J. Falco, K. Scarfone, et al., Guide to industrial control systems (ics) security, NIST special publication 800 (2011) 16–16.
[2] Y. Cherdantseva, P. Burnap, A. Blyth, P. Eden, K. Jones, H. Soulsby, K. Stoddart, A review of cyber security risk assessment methods for scada systems, Computers & security 56 (2016) 1–27.
[3] D. Bhamare, M. Zolanvari, A. Erbad, R. Jain, K. Khan, N. Meskin, Cybersecurity for industrial control systems: A survey, computers & security 89 (2020) 101677.
[4] C. Alcaraz, J. Lopez, Analysis of requirements for critical control systems, International journal of critical infrastructure protection 5 (2012) 137–145.
[5] P. Huitsing, R. Chandia, M. Papa, S. Shenoi, Attack taxonomies for the modbus protocols, International Journal of Critical Infrastructure Protection 1 (2008) 37–44.
[6] N. Goldenberg, A. Wool, Accurate modeling of modbus/tcp for intrusion detection in scada systems, international journal of critical infrastructure protection 6 (2013) 63–75.
[7] N. Erez, A. Wool, Control variable classification, modeling and anomaly detection in modbus/tcp scada systems, International Journal of Critical Infrastructure Protection 10 (2015) 59–70.
[8] R. R. R. Barbosa, R. Sadre, A. Pras, Exploiting traffic periodicity in industrial control networks, International journal of critical infrastructure protection 13 (2016) 52–62.
[9] T. Li, Y. Wang, C. Zou, Y. Tian, L. Zhou, Y. Zhu, Research on dos attack detection method of modbus tcp in openplc, Journal of Computer and Communications 9 (2021) 73–90.
[10] R. Sommer, V. Paxson, Outside the closed world: On using machine learning for network intrusion detection, in: 2010 IEEE symposium on security and privacy, IEEE, 2010, pp. 305–316.
[11] M. A. Teixeira, M. Zolanvari, K. M. Khan, R. Jain, N. Meskin, Flow-based intrusion detection algorithm for supervisory control and data acquisition systems: A real-time approach, IET Cyber-Physical Systems: Theory & Applications 6 (2021) 178–191.
[12] M. Kaouk, J.-M. Flaus, M.-L. Potet, R. Groz, A review of intrusion detection systems for industrial control systems, in: 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT), IEEE, 2019, pp. 1699–1704.

[13] M. A. Umer, K. N. Junejo, M. T. Jilani, A. P. Mathur, Machine learning for intrusion detection in industrial control systems: Applications, challenges, and recommendations, International Journal of Critical Infrastructure Protection 38 (2022) 100516.
[14] S. D. Anton, S. Kanoor, D. Fraunholz, H. D. Schotten, Evaluation of machine learning-based anomaly detection algorithms on an industrial modbus/tcp data set, in: Proceedings of the 13th international conference on availability, reliability and security, 2018, pp. 1–9.
[15] B. Phillips, E. Gamess, S. Krishnaprasad, An evaluation of machine learning-based anomaly detection in a scada system using the modbus protocol, in: Proceedings of the 2020 ACM southeast conference, 2020, pp. 188–196.
[16] S. Russo, C. Zanasi, I. Marasco, M. Colajanni, Autoencoder-based solution for intrusion detection in industrial control system, in: Science and Information Conference, Springer, 2024, pp. 530–543.
[17] S. Russo, C. Zanasi, I. Marasco, Feature extraction for anomaly detection in industrial control systems, Proceedings of the ITASEC (2024).
[18] I. Marasco, K. Chichifoi, S. Russo, C. Zanasi, Enhancing training time and sustainability in intrusion detection systems on machine learning (2025).
[19] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, F. Roli, Evasion attacks against machine learning at test time, in: Joint European conference on machine learning and knowledge discovery in databases, Springer, 2013, pp. 387–402.
[20] B. Biggio, F. Roli, Wild patterns: Ten years after the rise of adversarial machine learning, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 2154–2156.
[21] M. J. De Lucia, C. Cotton, Adversarial machine learning for cyber security, Journal of Information Systems Applied Research 12 (2019) 26.
[22] F. Pierazzi, F. Pendlebury, J. Cortellazzi, L. Cavallaro, Intriguing properties of adversarial ml attacks in the problem space, in: 2020 IEEE symposium on security and privacy (SP), IEEE, 2020, pp. 1332–1349.
[23] G. Zizzo, C. Hankin, S. Maffeis, K. Jones, 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, 2020, pp. 899–910.
[24] E. Anthi, L. Williams, M. Rhode, P. Burnap, A. Wedgbury, Adversarial attacks on machine learning cybersecurity defences in industrial control systems, Journal of Information Security and Applications 58 (2021) 102717.
[25] A. Mustafa, M. T. Khan, M. A. Umer, Z. Masood, C. M. Ahmed, Adversarial sample generation for anomaly detection in industrial control systems, in: Proceedings of the 1st Workshop on Modeling and Verification for Secure and Performant Cyber-Physical Systems, 2025, pp. 1–7.
[26] K. Boakye-Boateng, A. A. Ghorbani, A. H. Lashkari, Securing substations with trust, risk posture, and multi-agent systems: A comprehensive approach, in: 2023 20th Annual International Conference on Privacy, Security and Trust (PST), IEEE, 2023, pp. 1–12.
[27] H. Kheddar, Y. Himeur, A. I. Awad, Deep transfer learning for intrusion detection in industrial control networks: A comprehensive review, Journal of Network and Computer Applications 220 (2023) 103760.
[28] A. Swales, et al., Open modbus/tcp specification, Schneider Electric 29 (1999) 19.
[29] V. M. Igure, S. A. Laughter, R. D. Williams, Security issues in scada networks, computers & security 25 (2006) 498–506.
[30] M. Du, F. Li, G. Zheng, V. Srikumar, Deeplog: Anomaly detection and diagnosis from system logs through deep learning, in: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, 2017, pp. 1285–1298.
[31] A. Khraisat, I. Gondal, P. Vamplew, J. Kamruzzaman, Survey of intrusion detection systems: techniques, datasets and challenges, Cybersecurity 2 (2019) 1–22.
[32] Y. Hu, A. Yang, H. Li, Y. Sun, L. Sun, A survey of intrusion detection on industrial control systems, International Journal of Distributed Sensor Networks 14 (2018) 1550147718794615.
[33] G. Apruzzese, P. Laskov, J. Schneider, Sok: Pragmatic assessment of machine learning for network

intrusion detection, in: 2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P), IEEE, 2023, pp. 592–614.
[34] G. Vormayr, J. Fabini, T. Zseby, Why are my flows different? a tutorial on flow exporters, IEEE Communications Surveys & Tutorials 22 (2020) 2064–2103.
[35] Y. Sekaran, T. Debnath, T. A. Assadi, S. D. Suvvari, S. Oswal, Using machine learning to detect abnormalities on modbus/tcp networks, in: Proceedings of the 4th international conference on information management & machine intelligence, 2022, pp. 1–6.
[36] H. Zhang, Y. Min, S. Liu, H. Tong, Y. Li, Z. Lv, Improve the security of industrial control system: A fine-grained classification method for dos attacks on modbus/tcp, Mobile Networks and Applications 28 (2023) 839–852.
[37] E. Amer, B. A. S. Al-rimy, S. El-Sappagh, Strengthening ics defense: Modbus-nfa behavior model for enhanced anomaly detection, Journal of Information Security and Applications 89 (2025) 103990.
[38] N. Papernot, P. McDaniel, A. Sinha, M. P. Wellman, Sok: Security and privacy in machine learning, in: 2018 IEEE European symposium on security and privacy (EuroS&P), IEEE, 2018, pp. 399–414.
[39] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, F. Roli, Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks, in: 28th USENIX security symposium (USENIX security 19), 2019, pp. 321–338.
[40] B. Biggio, B. Nelson, P. Laskov, Poisoning attacks against support vector machines, arXiv preprint arXiv:1206.6389 (2012).
[41] H. Kettani, P. Wainwright, On the top threats to cyber systems, in: 2019 IEEE 2nd international conference on information and computer technologies (ICICT), IEEE, 2019, pp. 175–179.
[42] G. Banga, Why is cybersecurity not a human-scale problem anymore?, Communications of the ACM 63 (2020) 30–34.
[43] I. Corona, G. Giacinto, F. Roli, Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues, Information sciences 239 (2013) 201–225.
[44] G. Apruzzese, M. Colajanni, M. Marchetti, Evaluating the effectiveness of adversarial attacks against botnet detectors, in: 2019 IEEE 18th International Symposium on Network Computing and Applications (NCA), IEEE, 2019, pp. 1–8.
[45] K. He, D. D. Kim, M. R. Asghar, Adversarial machine learning for network intrusion detection systems: A comprehensive survey, IEEE Communications Surveys & Tutorials 25 (2023) 538–566.
[46] Y. Jia, J. Wang, C. M. Poskitt, S. Chattopadhyay, J. Sun, Y. Chen, Adversarial attacks and mitigation for anomaly detectors of cyber-physical systems, International Journal of Critical Infrastructure Protection 34 (2021) 100452.
[47] Y. M. Khaw, A. A. Jahromi, M. F. Arani, D. Kundur, Evasive attacks against autoencoder-based cyberattack detection systems in power systems, Energy and AI 17 (2024) 100381.
[48] Y. Liu, L. Xu, S. Yang, D. Zhao, X. Li, Adversarial sample attacks and defenses based on lstm-ed in industrial control systems, Computers & Security 140 (2024) 103750.
[49] S. S. A. Naqvi, C. Zhou, P. Xu, Y. Li, J. Jiashu, M. Uzair, Adversarial feature generation for ml-based intrusion detection in the petrochemical industry, Journal of Information Security and Applications 94 (2025) 104215.
[50] I. J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, arXiv preprint arXiv:1412.6572 (2014).
[51] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, arXiv preprint arXiv:1706.06083 (2017).
[52] G. Apruzzese, M. Andreolini, L. Ferretti, M. Marchetti, M. Colajanni, Modeling realistic adversarial attacks against network intrusion detection systems, Digital Threats: Research and Practice (DTRAP) 3 (2022) 1–19.
[53] G. Apruzzese, H. S. Anderson, S. Dambra, D. Freeman, F. Pierazzi, K. Roundy, "real attackers don't compute gradients": bridging the gap between adversarial ml research and practice, in: 2023 IEEE conference on secure and trustworthy machine learning (SaTML), IEEE, 2023, pp. 339–364.
[54] D. Wu, B. Fang, J. Wang, Q. Liu, X. Cui, Evading machine learning botnet detection models via deep

reinforcement learning, in: ICC 2019-2019 IEEE International Conference on Communications (ICC), IEEE, 2019, pp. 1–6.

[55] D. Han, Z. Wang, Y. Zhong, W. Chen, J. Yang, S. Lu, X. Shi, X. Yin, Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors, IEEE Journal on Selected Areas in Communications 39 (2021) 2632–2647.

[56] N. Wang, Y. Chen, Y. Xiao, Y. Hu, W. Lou, Y. T. Hou, Manda: On adversarial example detection for network intrusion detection system, IEEE Transactions on Dependable and Secure Computing 20 (2022) 1139–1153.

[57] G. Apruzzese, A. Fass, F. Pierazzi, When adversarial perturbations meet concept drift: an exploratory analysis on ml-nids, in: Proceedings of the 2024 Workshop on Artificial Intelligence and Security, 2024, pp. 149–160.

[58] G. Apruzzese, M. Colajanni, Evading botnet detectors based on flows and random forest with adversarial samples, in: 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), IEEE, 2018, pp. 1–8.

[59] G. Engelen, V. Rimmer, W. Joosen, Troubleshooting an intrusion detection dataset: the cicids2017 case study, in: 2021 IEEE Security and Privacy Workshops (SPW), IEEE, 2021, pp. 7–12.

[60] L. Liu, G. Engelen, T. Lynar, D. Essam, W. Joosen, Error prevalence in nids datasets: A case study on cic-ids-2017 and cse-cic-ids-2018, in: 2022 IEEE Conference on Communications and Network Security (CNS), IEEE, 2022, pp. 254–262.

[61] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, K. Rieck, Dos and don'ts of machine learning in computer security, in: 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 3971–3988.

[62] P. Owezarski, Investigating adversarial attacks against random forest-based network attack detection systems, in: NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2023, pp. 1–6.

[63] A. Venturi, D. Galli, D. Stabili, M. Marchetti, et al., Hardening machine learning based network intrusion detection systems with synthetic netflows, in: CEUR WORKSHOP PROCEEDINGS, volume 3731, CEUR-WS, 2024.