When Adversarial Perturbations meet Concept Drift: an Exploratory Analysis on ML-NIDS

Giovanni Apruzzese
giovanni.apruzzese@uni.li
Liechtenstein Business School
University of Liechtenstein
Vaduz, Liechtenstein

Aurore Fass
fass@cispa.de
CISPA Helmholtz Center for
Information Security
Saarbruecken, Germany

Fabio Pierazzi
fabio.pierazzi@kcl.ac.uk
Department of Informatics
King's College London
London, United Kingdom

Abstract

We scrutinize the effects of "blind" adversarial perturbations against machine learning (ML)-based network intrusion detection systems (NIDS) affected by concept drift. There may be cases in which a real attacker – unable to access and hence unaware that the ML-NIDS is weakened by concept drift – attempts to evade the ML-NIDS with data perturbations. It is currently unknown if the cumulative effect of such adversarial perturbations and concept drift leads to a greater or lower impact on ML-NIDS. In this "open problem" paper, we seek to investigate this unusual, but realistic, setting—we are not interested in perfect knowledge attackers.

We begin by retrieving a *publicly available* dataset of documented network traces captured in a real, large (>300 hosts) organization. Overall, these traces include several years of raw traffic packets—both benign and malicious. Then, we adversarially manipulate malicious packets with problem-space perturbations, representing a *physically realizable attack*. Finally, we carry out the *first exploratory analysis* focused on comparing the effects of our "adversarial examples" with their respective unperturbed malicious variants in concept-drift scenarios. Through two case studies (a "short-term" one of 8 days; and a "long-term" one of 4 years) encompassing 48 detector variants, we find that, although our perturbations induce a lower detection rate in concept-drift scenarios, some perturbations yield adverse effects for the attacker in intriguing use cases. Overall, our study shows that the topics we covered are still an open problem which require a re-assessment from future research.

CCS Concepts

• **Security and privacy → Intrusion detection systems**; • **Computing methodologies → Machine learning**.

Keywords

network intrusion detection, adversarial example, machine learning, mcfp, ctu13, data drift, distribution shift, temporal evaluation

ACM Reference Format:
Giovanni Apruzzese, Aurore Fass, and Fabio Pierazzi. 2024. When Adversarial Perturbations meet Concept Drift: an Exploratory Analysis on ML-NIDS. In *Proceedings of the 2024 Workshop on Artificial Intelligence and Security (AISec '24), October 14–18, 2024, Salt Lake City, UT, USA*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3689932.3694757

1 Introduction

The protection of modern network infrastructures is typically reliant on Network Intrusion Detection Systems (NIDS) [10, 63]. To give NIDS a chance against the constantly mutating threat landscape, state-of-the-art defenses now rely on Machine Learning (ML) methods [11, 33]. Unfortunately, the integration of ML in NIDS is not exempt from problems: first, the intrinsic evolution of the network environment ("concept drift"), which leads to performance degradation over time [35]; second, the susceptibility of ML models to tiny perturbations in the input data ("adversarial examples"), which may enable an attacker to bypass the defense [15]. Taken individually, both of these phenomena have been widely investigated by prior literature [5, 39]. However, we wonder: *what happens if an "unaware" attacker attempts to adversarially evade an ML-NIDS impacted by concept drift?* Perhaps surprisingly, no work (background in §2.1) simultaneously considers the effects of (realizable) adversarial perturbations against ML-NIDS trained on "outdated" datapoints. In these circumstances, which resemble real-world deployments of ML-NIDS [10, 75], is concept drift an ally of the attacker or of the defender? We seek to shed light on this question.

The first challenge we have to overcome to pursue our quest is **finding a proper dataset**. Contrarily to traditional malware detection repositories (e.g., `AndroZoo` [52]) – containing constantly updated and publicly available historical data – the NIDS context is much more nuanced: every network is unique, and publicly available data is scarce [11]. For instance, the evaluation of a notable work on concept drift [5] considers the `CICIDS17` [61] dataset, which contains only 5 days of traffic. After searching the current ecosystem of open datasets for ML-NIDS, we found a solution in the *datasets from the Malware Capture Facility Project* (`MCFP` [1]). We discuss our choice (as well as some pitfalls of prior work) in §3.

The second challenge is realizing **realistic evasion attacks**. It is well known [15] that ML models can be evaded by adversaries with full knowledge of the ML model [31], or with the possibility of querying the target ML model [78]. However, such knowledge / capabilities are not free to acquire for real attackers [6] (e.g., only sysadmins can observe the output of an NIDS [8]), who may opt for different (and potentially less-accurate) strategies. Moreover, abundant prior work (e.g., [36, 77]) on adversarial ML attacks against ML-NIDS relies on feature-space perturbations, i.e., the manipulation occurs *after* the data has been preprocessed by the NIDS, which is not very realistic [8] and may also reflect physically unrealizable perturbations [53]. Hence, for our attacks *we operate on the raw* *network packets*, which we manipulate by introducing random bytes of junk data—doable by any attacker who can control the network communications of some hosts (threat model in §2.2). We explain our procedure and discuss its real-world validity in §4.

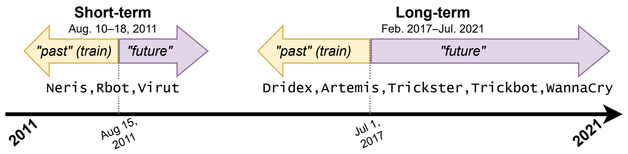


Fig. 1: Overview of our evaluation. We carry out two case studies ("short-term" and "long-term") using real-network data from MCFP [1] captured between 2011 and 2021, and pertaining to various malicious classes as well as benign traffic. We craft adversarial perturbations in the problem space, manipulating the datapoints generated in the "future", and compare their impact to non-adversarial datapoints.

After having overcome these challenges, we carry out an exploratory analysis, which revolves around **two case studies**—sharing a similar design but spanning over vastly different timespans (see Fig. 1). Specifically, both case studies focus on the problem of malicious NetFlow detection, and entail assessing the performance of various ML-NIDS (employing different ML algorithms and architectures) in adversarial and non-adversarial settings. The first case study spans over just one week (August 10–18th, 2011), whereas the second case study spans over four years (from 2017 to 2021). In both case studies, we assess the performance of our considered ML-NIDS at different points in time. After training ML-NIDS on some "past" data, we assess the impact of concept drift by measuring their performance on "future" data—thereby validating our testbed (in terms of data and baselines). Then, we focus on *malicious* NetFlows, and we scrutinize whether the application of adversarial perturbations (in the problem space) makes such NetFlows more or less likely to evade the ML-NIDS—compared to their unperturbed (but still malicious) variants. We also consider an attack-agnostic adversarial ML defense suitable for NetFlow-based ML-NIDS [7] but whose effectiveness in our setting is still unexplored. We repeat all our experiments 50 times, ensuring statistical robustness of our results. Intriguingly, our results (discussed in §5) show that some adversarial perturbations can be detrimental to the attacker if the ML-NIDS is under concept drift, confirmed by the resulting adversarial NetFlows being easier to detect (we try to interpret these unexpected phenomena with low-level analyses in §5.3).

Contributions. This paper sheds light on a problem that has never been investigated before in the ML-NIDS context: the combination of realistic (blind, realizable) adversarial perturbations with concept drift. To this end, we:

- pinpoint an open-source (and documented) *dataset that can be used for concept-drift assessments* in ML-NIDS (and which has been overlooked by most research);
- craft *problem-space adversarial perturbations* by manipulating raw network traffic (and share our implementation, which works on any PCAP), simulating a simple and feasible attack;
- investigate the extent to which ML-NIDS are *statistically significantly affected* by realistic adversarial perturbations in concept-drift contexts—showcasing why we tackle an "open problem".

We also derive recommendations (§6) for future research, and we publicly release our resources [3], which also include benchmarking results and additional descriptions of our workflow and methods.

2 Preliminaries and Scope

To setup the stage for our contribution, we first position our paper within existing literature (§2.1); then we present our envisioned and realistic threat model (§2.2).

2.1 Background and Related Work

Our paper lies at the intersection of three popular research areas of cyber security: NIDS, ML for security, and security of ML. In what follows, we briefly summarise all of these.

ML-NIDS. Since complete prevention of cyberattacks is unattainable, intrusion *detection* systems (IDS) represent the primary line of defense of modern organizations against cyber threats [10]. These systems perform their detection via rules [67] (which cannot capture zero-day attacks) and/or data-driven heuristics—such as those based on machine learning [63]. An IDS can analyse various information, which can be captured either at the host- or network-level [11, 20]. *In this paper, we focus on machine-learning–based network intrusion detection systems* (ML-NIDS), for which numerous papers exist (whose evaluations have been done both on synthetic [61] and real-world [28] data). Despite some shortcomings (e.g., false alarms [4]) there is evidence that ML is being integrated in operational platforms [58], and practitioners believe the combination of rules and ML to be the best way to address network security [44]. Unfortunately, ML methods are affected by two security-noteworthy issues [10]: their susceptibility to "concept drift" [14] and to "adversarial ML attacks" [15]—both of which are exacerbated in a network context (as pointed out in, e.g., [11, 17]).

Concept Drift. Abundant prior efforts have shown that the performance of ML models deteriorates over time [38]. This phenomenon stems from the ML model being "overly-reliant" on its training data: if the data seen at inference differs substantially from that seen by the ML model during its learning phase, then its output may not be correct [10]. Unfortunately, modern networks are constantly-mutating ecosystems—from both the "benign" (e.g., devices may be added or removed) and "malicious" (attackers refine their tactics) perspective [11, 42]. Extant cybersecurity literature mostly investigated concept drift in malware detection contexts [14, 16, 62], with some recent efforts considering this (still open) problem also for ML-NIDS [5, 46, 70, 72, 75]. Despite their undeniable contributions, however, we argue that most related research has shortcomings due to an intrinsic *limited scope of the adopted testbed*, which hardly resembles realistic use cases (we motivate this statement in §3.1).

Attacks vs ML-NIDS. Security assessments of IDS in adversarial settings are not new [23, 31]. A large body of recent literature specifically focused on the vulnerability of ML-NIDS to "adversarial ML attacks" [8], wherein the goal is to mislead the ML model through tiny "data perturbations". These ML-intrinsic vulnerabilities can affect the ML model either before or after its deployment[15]: *we focus on evasion attacks at inference* (we are not interested in "poisoning" ML-NIDS [21, 59, 60]). Early work in this domain (e.g., [9]) considered attacks whose perturbations were crafted in the "feature space": as pointed out in [53], such a procedure may result in attacks that are not physically realizable—unless the adversary can directly tamper with the ML-NIDS [8] (which is a not very realistic scenario [6] that is outside our scope). Some works overcome such a limitation by applying the perturbations in the "problem space", i.e., by manipulating the network packets generated by the attacker-controlled hosts [30, 71, 73]. Nevertheless, a recent work [6] highlighted that most adversarial ML research considers powerful attackers (in terms of knowledge and capabilities), making the corresponding threat models difficult to enact in practice.

Research Gap. No prior work specifically investigated the effects of blind adversarial perturbations against ML-NIDS under concept drift in a realistic setting—which is our goal.

The closest paper we could find is the ACSAC'22 work by Wang [72]. However: *(i)* its concept-drift evaluation is affected by the *data limitations* outlined in §3.1; *(ii)* the attacker is assumed to be able to *query the ML-NIDS*, which is unrealistic [6]; *(iii)* the perturbations are crafted in the *feature space*, potentially leading to unrealizable attacks [53]. Hence, to the best of our knowledge, this is still an open problem—which we seek to cast some light upon.

2.2 Threat Model (adv. perturbations + concept drift)

Our analysis is rooted on a realistic threat model, which depicts a constrained attacker that must evade a state-of-the-art ML-NIDS. A schematic representation of our threat model is provided in Fig. 2.

Target System. We consider an ML-NIDS deployed at the perimeter of a "Class B", medium-to-large network [2] (i.e., [256–65k] hosts). The ML-NIDS integrates some ML models, which are collectively trained to identify (malicious) network traffic pertaining to $M + 1$ classes—which can be seen as either a binary- or multi-classification ML problem (having M malicious classes, and one benign class). The ML-NIDS accepts input in the form of network flows (NetFlow; see Appendix A), which is a practical datatype [11, 50, 55]. The ML-NIDS may integrate additional analytical elements (e.g., signature-based detectors of payload [67]) but these are outside our scope. The owners of the ML-NIDS have verified that the ML models exhibit high true-negative rate (tnr) and high true-positive rate (tpr) before its deployment, and hence expect the ML-NIDS to "work well" to identify "known" threats (i.e., whose datapoints can be attributed to one among the M classes) after its deployment. The owners, however, are unaware of potential concept drift.

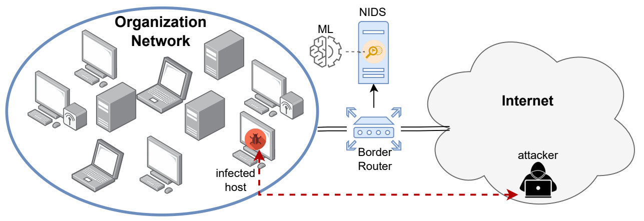


Fig. 2: Threat Model. The attacker is outside the organization's network, cannot access the NIDS, and only knows that an ML model analyzes network traffic.

Envisioned Attacker. The adversary has obtained control of some (low privileged) hosts within the organization's network—via, e.g., exploits or accidental infection. The attacker *wants to remain undetected*, so that they can achieve their true objective (e.g., exfiltration or sabotage [68]). The attacker is, however, heavily constrained—from a traditional "adversarial ML" perspective.

- **Knowledge.** The attacker has limited knowledge of the ML-NIDS. They only know that *(i)* some ML model analyzes network data; and that *(ii)* such ML model has "seen" datapoints of the piece of malware used to control the host. However, the attacker does not know: the details/configuration of the ML model; the exact type of analyzed data (i.e., "features"); the training set; and whether the ML-NIDS is affected by concept drift.
- **Capabilities.** The attacker can fully control their infected hosts. However, the attacker has no power on the ML-NIDS: they cannot observe its output (i.e., for "query-based" strategies [72]), and cannot manipulate the traffic beyond the controlled host (i.e., for "feature-space" perturbations [9]). Moreover, the attacker can control only a limited number of hosts, hence the attacker cannot create a "surrogate" ML-NIDS [65] by passively capturing traffic (which would never be representative of the entire organization).

In this context,[1] the attacker has *two strategies*: *(a)* do nothing—the attacker may "hope" that the ML-NIDS will not be able to detect their presence (e.g., due to concept drift, or because the alarms are not triaged [66]); or *(b)* try to evade the ML-NIDS—however, the realistic constraints prevent launching "guaranteed" adversarial ML attacks [6], hence the attacker can only introduce blind perturbations[2] which may (or may not!) help in remaining concealed.

A real-attacker's dilemma. It is unknown whether it is beneficial to introduce blind adversarial perturbations in an attempt to bypass ML-NIDS affected by concept drift. The attacker must make a choice—potentially one which may be detrimental to their goal. We seek to explore this dilemma—which is relevant for real-world ML-NIDS: these are well known to be subject to concept drift and are also deployed in adversarial environments [10].

3 Data Preparation (a "how to" guide)

As our first contribution, we elucidate the challenges that entail fair and realistic assessments of concept drift in ML-NIDS from a data perspective (§3.1) and then describe our identified "solution," i.e., the MCFP dataset (§3.2). Finally, we explain how we use MCFP (§3.3).

3.1 Challenge: Finding the right data

Carrying out assessments on "concept drift" is not trivial in research, due to the lack of appropriate datasets. Importantly: abundant prior work has expressed concerns on the overall utility of well-known publicly available datasets for NIDS-related research (e.g., [10, 19, 32, 37]), but no work has specifically analysed their suitability for concept-drift assessments. We will now do such an analysis.

Data Requirements. As highlighted in [11], investigating the effects of concept drift on ML-NIDS requires a dataset that can realistically represent a concept-drift setting. In practice, this necessitates that *(i)* the data captures a "long" timeframe; *(ii)* the malicious data reflects "naturally-occurring" phenomena; and *(iii)* the data is "unambiguously" labeled. The first requirement serves to assess the performance over time; the second serves to avoid biased evaluations in which malicious datapoints are artificially created and would not naturally occur; the third serves to ascertain that potential misclassifications are fairly validated. While these threefold requirements are relatively easy to fulfill for, e.g., malware detection (due to the existence of publicly available and constantly maintained repositories containing "historical" malware [14, 45, 52, 74]), this is

[1] **Remark:** according to [6], the ML-NIDS is an "invisible ML system" for our attacker. We stress that our scenario is sensible and portrays a much weaker (but hence more likely!) attacker than those envisioned in most prior works on ML-NIDS [8, 31].

[2] Prior work (*not on concept drift*) found "blind" perturbations can be effective [9, 48].

not the case for ML-NIDS. Indeed, most currently available datasets *do not allow for satisfactory concept-drift evaluations.*

Data Pitfalls. Let us discuss existing publicly available datasets for ML-NIDS in light of prior research on concept drift. We begin with the NSL-KDD dataset (used in [46]), which is well known to present flaws that do not make it representative of real-world networks [10]. Then, we mention CICIDS17 (used in [72]) and its extension CICIDS18 (used in [75]): both of which have been recently criticized for being intrinsically flawed [27, 37] (notably, [5] uses the fixed version of CICIDS17); regardless, they include data created via simulations and spanning over only 5 days—which is not enough to gauge the effects of concept drift on a realistic deployment of an ML-NIDS. Next, there is UNSWNB15 (used in [41]), which only contains 15 hours of traffic. Even the dataset by Pahl et al. [51] (used in [70]) and Kitsune (used in [72]) have barely 1 day of capture. A noteworthy mention is UGR16 (used in [22]), with traffic spanning over 100 days: the issue is that the labeling is not consistent,[3] and is mostly appropriate for "anomaly detection" (which does not necessarily pertain to cyber threats, since an anomaly is not necessarily an intrusion event [12]). A similar issue also affects the Kyoto2006 dataset (used in [25, 29]), which is collected over 15 years but for which there is no validated ground truth. Some works (e.g., [40, 72]) also attempt to "mix" datasets captured from substantially different networks, which is considered to be an unreliable procedure [11].

Disclaimer. We are not "pointing the finger," and we do not seek to invalidate prior work. On the contrary, we want to reflect on the current state of concept-drift evaluations in ML-NIDS—with the ultimate goal of improving this research domain.

3.2 A (open source) solution: the MCFP dataset

After surveying the landscape of public datasets for ML-NIDS, we found a solution[4] which enables comprehensive assessments of concept drift: the data from Malware Capture Facility Project (MCFP) [1], which extends the well-known CTU13 (created in 2014 [28]).

Characteristics. The MCFP is a collection of packet captures (PCAP) provided with accurate ground-truth information of (among others): the infected hosts, the piece of malware, as well as the start and end time of the infection. Importantly, the malicious packets are generated by "contemporary" malware: the creators of MCFP infect their hosts with pieces of recent malware (e.g., WannaCry was used in 2017) to study their behavior (for comparison, CICIDS17 artificially created malicious traffic by using offensive techniques popular many years before[5]). Moreover, MCFP is captured in a real university campus, and the "benign" traffic pertains to hundreds (>300) of hosts (for comparison, CICIDS17 and Kitsune only entail a network with a dozen hosts). Notably, MCFP is built on top of the well-known CTU13 dataset [28] (which spans over a single week, starting from August 10th, 2011); however, MCFP contains traffic up to 2021. Indeed, among the strongest points of MCFP is that it contains up-to-date *benign and malicious* data—the latter encompassing a number of network-related malware (such as WannaCry or Trickbot) whose traffic is captured *over long timespans* (even years!). Finally, having been collected by the same (set of) creators, it is *consistent.*[6]

Takeaway. The MCFP is a suitable solution for assessments of ML-NIDS under concept drift. It is *large*, entails *long* timespans, *ground truth* is provided, and includes *various* types of benign and malicious (entailing *recent* attacks) traffic in PCAP format.

3.3 Practically using MCFP data (our testbed)

We explain how we used MCFP to carry out our exploratory analysis.

Design Goals. Recall that we are not only interested in assessing the impact of concept drift on ML-NIDS, but also in investigating the effects of adversarial perturbations crafted with the intent of having malicious NetFlows be classified as benign. Hence, we need (labeled) malicious and benign NetFlows. Furthermore, since we consider problem-space attacks, we must manipulate the PCAP (which is provided in MCFP). Moreover, we also want the analysis to encompass a "large" number of NetFlows (otherwise, the results would be questionable). Finally, for a fair comparison, we must ensure that for any considered malicious class, the ML-NIDS has seen NetFlows of such a class during its training phase:[7] this is *necessary*, because misclassifications of NetFlows stemming from "novel" attacks cannot be attributed to concept drift (it is an "out-of-distribution" case [35]).

Benign Data. We first retrieve *all benign data* available in MCFP, which is summarized in Table 5 (in the Appendix); detailed information on each PCAP is provided by the source (reachable by "clicking" on the links in Table 5). Altogether, we take benign traffic from the original CTU13 (related to both "background" and "active" communications), as well as more "recent" traffic. Besides the size of the corresponding PCAP, we also report the number of NetFlows generated by processing the PCAP via Argus [56]. We consider Argus for two reasons: **(1)** it is used by the creators of CTU13 to create and label [28] the NetFlows—which is necessary to ensure that our labeling is correct;[8] **(2)** because it yields practical ML-NIDS (as shown in [11, 28]) and has no known flaws (e.g., CICFlowMeter has issues [27]). Overall, our benign datasets span over 7 years, and contain millions of NetFlows—allowing for a sound analysis.

Malicious Data. Then, we selectively inspect the various PCAP pertaining to malicious traffic. We found that some attacks do not allow for a sound analysis (e.g., short timeframe, or small number of NetFlows). Ultimately,[9] we considered 65 PCAP traces pertaining to 8 network-related malware, reported in the Appendix in Table 6 (having the same structure as Table 5). Specifically, three

[3] Furthermore, UGR16 does not have PCAP, preventing problem-space perturbations.

[4] We do not claim to have created MCFP, nor that this dataset is the only one suitable for our scope; we are, however, underscoring its relevance for future work.

[5] E.g., the CICIDS17 (from 2017) uses Heartbleed which became popular in 2014.

[6] Despite our search, we are not aware of any other open-source dataset that has the same characteristics as MCFP and that has more recent traffic. Surprisingly, the only prior work on concept drift we found that considered a similar testbed was [49] (albeit we are unsure how the temporal aspect was accounted for in [49]; see §6)

[7] **Formally**, let M be the set of malicious classes known by an ML model $\mathcal{C}$; let $m \in M$ be a class, and let m_t be the NetFlows associated to m included in the *training* set of $\mathcal{C}$. We want to test the performance of $\mathcal{C}$ on NetFlows m_i, i.e., belonging to class m but analysed during the *inference* stage of $\mathcal{C}$ (with $m_i \neq m_t$). Therefore, in this paper we focus on *supervised* ML (and not on unsupervised ML methods [26]).

[8] Recall that we will apply our perturbations in the problem space, i.e., by manipulating the raw traffic. Hence, we must generate, and then manually label, the corresponding feature representation (i.e., the Network Flows) to train and test our ML-NIDS.

[9] We provide in our repository [3] a supplementary document describing the detailed procedure we followed to derive our considered malicious classes.

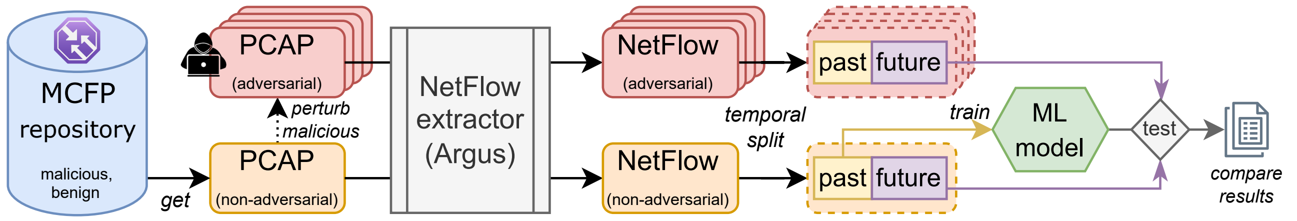


Fig. 3: Experimental Workflow. We get benign and malicious PCAP from MCFP and manipulate *only* the malicious traces—yielding problem-space adversarial perturbations. Next, we take all PCAPs and extract (and label) the corresponding NetFlows. Finally, we train ML models (on both benign and malicious "past" data) and test their effectiveness on "future" data (benign, malicious, and adversarial) in a concept-drift setting. We repeat our experiments 50× to ensure statistically sound comparisons of our results.

are from CTU13 (Neris, Rbot, Virut), while five are more recent (Artemis, Trickbot, Trickster, Wannacry, Dridex). Overall, our selection allows for a comprehensive analysis, given that we have over 40k NetFlows per class. We consider both "recent" and "older" attacks because we carry out two case studies (see Fig. 1), discussed in the next section.

4 Experiment Setup & Implementation

To prepare our main contribution, we now craft our adversarial perturbations (§4.1) and develop the ML models (§4.2) used in our exploratory analysis (§4.3). Our workflow is shown in Fig. 3.

4.1 "Blind" Adv. Perturbations on Raw Traffic

In crafting our "adversarial examples", we are inspired by prior works [9, 30, 53, 73]. We do not aim to assess "novel" attacks.

Context. Recall that our attacker (§2.2) has limited knowledge and capabilities w.r.t. the ML-NIDS. However, the attacker can control their infected hosts and induce them to generate altered network communications—but without any assurance that such changes help evasion. The attacker is, however, aware of extant research. Prior work [9] showed that NetFlow ML-classifiers exhibited a lower detection rate against datapoints having tiny alterations (applied in the feature space) in the exchanged bytes. Such an effect can be reproduced in the problem space by "blindly" appending junk data to network packets [73]. Practically, we assume that the attacker adds padding data to UDP packets, as well as to TCP packets with the PSH flag (some TCP packets do not admit padding). Indeed, while operating in the problem space we need to ensure that the resulting perturbations *(i)* do not break the packets' malicious semantics/functionality [24], and *(ii)* do not violate domain constraints [53]. Our choice ensures compliance with this twofold requirement, while aligning with our envisioned attacker.

Remark: we focus on assessments of *malicious* (adversarial and non-adversarial) datapoints for evasion. We do not manipulate any benign data (but we will use these for validation purposes).

Problem-space Manipulations. To realize our perturbations, we first take the malicious PCAP traces (Table 6 in the Appendix). Then, for each PCAP trace, we iterate over all the packets contained therein and create *four* "adversarial" PCAP traces—each by selectively manipulating a subset of its packets. Specifically:

- we create one adversarial trace by manipulating *only* UDP packets, by adding a small padding of [1–100] random bytes to their payload (this range is to align with the realistic setting of [9]);
- then, we create another adversarial trace by manipulating *only* TCP packets with the PSH flag active, also by adding a small padding of [1–100] random bytes to their payload;
- finally, we repeat the previous two steps, creating two "secondary" adversarial traces (one for UDP and one for TCP packets with the PSH flag) to mitigate the potential bias introduced by the random padding and provide more statistically robust results.

Every other packet is appended as-is to each adversarial PCAP trace. While applying the manipulations, we also: *(i)* ensure that each packet does not exceed its maximum length—in which case, the packet will be filled until the end; and *(ii)* recreate the checksum. These operations are done via the *scapy* library (used also in [30, 34]), and can be carried out by our envisioned attacker (§2.2). Finally, and crucially, our problem-space perturbations will transfer [53] to the "feature space" since our ML-NIDS use NetFlows [69] which include features that depend, among others, on the amount of data transmitted by the two endpoints (refer to Appendix A).

Open source. Few works assess ML-NIDS against adversarial perturbations in the problem space (§2.1), which requires careful operations on PCAP traces. To spearhead future research, we release the code of our "Packet Modifier" [3], which can be used by future work to assess different perturbations (by changing, e.g., the padding, the types of packets, or even consider benign traffic).

4.2 Preprocessing and Machine Learning

Having crafted our (adversarial and non-adversarial) PCAP traces, we must generate (and label) the NetFlows (i.e., the feature representation of our traces) to prepare them for our ML experiments.

Annotation. We extract the NetFlows from our raw PCAP traces with Argus [56] (see §3.2); the Argus config is in our repo [3]. Notably, while processing our adversarially-manipulated PCAP traces, Argus does not raise any errors—thereby confirming *(i)* the correctness of our procedure and *(ii)* that the resulting NetFlows are not "flawed"; of course, the output of Argus for our adversarial PCAP traces is different from the one for the corresponding non-adversarial ones (which is expected [69]). Next, we must assign the ground truth to each NetFlow. We rely on the official documentation of MCFP (and of CTU13). Specifically, we first label the NetFlows for the data corresponding to CTU13; then, to validate our procedure, we compare our NetFlows with those provided in CTU13 (this dataset is provided both as raw PCAP and as preprocessed NetFlows): we appreciate that there is an almost perfect match (we manually correct inconsistencies—for which we found no documentation in CTU13). Then, we label the NetFlows for all other PCAP traces by following the respective documentation (there are no labeled NetFlows available for these PCAP traces); we apply the same labeling logic for the original and adversarial NetFlows.

Threat to validity: We acknowledge some labeling errors may be present, but we adhered to the source documentation. Nevertheless, even practitioners admit to label coarsely [18, 66].

ML Detectors. For our exploratory analysis, we follow the best practices of prior work on "pragmatic assessment" of ML-NIDS [11]. First, to enable a broad analysis, we: *(a)* consider *two ML algorithms* which are known to provide good results on NetFlow classification [11, 50, 73]: Random Forest (RF) and Histogram-Gradient Boosting (HGB). Then, we *(b)* consider *various detection architectures*: a "full" binary classifier, treating all malicious NetFlows as a single malicious class; and an "ensemble" of binary classifiers, each focusing on a specific malicious class (more details in §5). All these classifiers analyze 38 common NetFlow values (e.g., exchanged bytes, packets, duration) and we avoid considering "inappropriate" features (e.g., IP addresses [13]) inducing overfitting; we also map the network ports to the IANA port types (as done in [11]). The complete details of our implementation (including the hyperparameters of our chosen ML models) are provided in our repository [3].

4.3 Case Studies (*malicious* NetFlow detection)

We explain how we orchestrate our resources to fulfill our objective.

Method. To carry out our exploratory analysis, we design two case studies (outlined in Fig. 1 in the Introduction), sharing similar properties but differing on the time perspective.

- Short-term Case Study (SCS), of one week (Aug. 10→18, 2011).
- Long-term Case Study (LCS), of four years (Feb. 2017→Jul 2021).

Specifically, we are inspired by [25, 52] and perform the following steps. **(1)** For each case study, we establish a *cutoff date*: the "past" data before such a date is used to develop our ML-NIDS, while the "future" data after such a date serves to test the ML-NIDS—both in adversarial and non-adversarial settings. **(2)** We *develop our ML-NIDS*: we consider the "past" data and perform a 70:30 train:test split (which is common [11, 43]); each ML-NIDS is trained on the same training data, and its performance (before deployment) is assessed on the same test data. We verify that the performance (high tpr and high tnr) justifies its deployment. **(3)** We implement an *adversarially hardened* variant for each ML-NIDS with the defense proposed in [7] (which has been found effective to harden NetFlow-based classifiers—but against feature-space perturbations). **(4)** We assess the *impact of concept drift* by measuring the performance on all ML-NIDS on "future" data. We expect a drop in the tpr and tnr w.r.t. those achieved on the "past" data. **(5)** We gauge our *adversarial perturbations* by testing all the ML-NIDS on "future" adversarial data. We will compare the evasiveness of non-adversarial (malicious) NetFlows with their adversarial variants. **(6)** To ensure *statistical robustness*, steps 2–5 are repeated 50 times—each by training all our detectors on a different (randomly drawn) subset of "past" data.[10] N.b.: such repetitions entail re-assessing each "new" detector against all variants (2xTCP, 2xUDP) of our adversarial data (as well as against the original malicious and benign NetFlows).

Details and Motivations. Here, we clarify some lingering questions related to our design choices. *[How to select the cutoff date?]* Given that we assume (see Footnote-7 and §2.2) that the ML-NIDS must be tested on data belonging to the malicious classes seen during training, our cutoff date is set to Aug. 15th, 2011 for SCS and Jul. 1st, 2017 for LCS. By observing Tables 6 in light of these dates, one can appreciate that all our considered malicious classes match our assumption. *[What about benign data?]* For a realistic assessment, we also train/test on benign data: for SCS, we use those in Table 5b; for LCS, we use the "background" of Table 5b and those in Table 5a (unfortunately, there are no "background" captures in MCFP beyond those in Table 5b).[11] To balance our datasets ("background" NetFlows are ≈10x "active"), we consider three background traces and drop 20% of their NetFlows (randomly chosen). Roughly, our ML-NIDS are trained on ~1M "background" NetFlows, and ~200K (for SCS) or ~400K (for LCS) "active" NetFlows. Regardless, since our priority is to investigate the simultaneous effects of concept drift and adversarial perturbations on ML-NIDS, the timespans of our case studies reflect the capture dates of *malicious* data (which is why LCS is reported to span across 2017–2021). *[Why two case studies?]* We consider two case studies to *(i)* enable a broad analysis, but also to *(ii)* increase the chances of witnessing concept drift. Most prior research (§3.1) only considered short timespans (e.g., one week [5]). The major issue of concept drift is that it is impossible to know a priori if, when, and how it will occur [11]. To the best of our knowledge, we are the first to consider this exact testbed for concept-drift assessments. Thus, even we are unsure of what we will find: considering both SCS and LCS increases the chances that our ML-NIDS will be affected by concept drift—enabling to study the combined effects of concept drift and adversarial perturbations.

5 Exploratory Analysis

We now delve into our primary contribution. We verify the presence of concept drift (§5.1), then gauge the impact of our adversarial perturbations (§5.2) and shed light on intriguing phenomena (§5.3).

Recap. Before we begin, let us emphasize some key points of our evaluation. Across our case studies (SCS and LCS), *we develop* 48 *ML-NIDS*: 20 for SCS and 28 for LCS. These numbers are given by: 2 ML algorithms (RF and HGB) × 2 hardening types (a "defense" and a "vanilla" variant) × 5 (for SCS) or 7 (for LCS) architectures. These architectures depend on the malicious classes considered in each case study (see Fig. 1), i.e., 3 for SCS (Neris,Rbot,Virut) and 5 for LCS (Artemis,Dridex,Trickbot,Trickster,Wannacry). For instance, in SCS, we have 5 architectures: a "full" binary classifier, trained and tested (for 50 times) on all three malicious classes (treated as a single malicious class) and on benign NetFlows; three "malware-specific" binary classifiers, trained and tested (for 50 times) on the specific malware class and on benign NetFlows; and an "ensemble" composed by all three malware-specific classifiers: the final output is determined in a LOR mechanism, wherein each malware-specific classifier analyzes any given NetFlow and, if at least one yields a "positive" prediction, the output is malicious (and benign otherwise). Our workflow ensures no "data snooping" (besides, we have no reason for this: our analysis is exploratory and we do not seek to "outperform" prior work!). These procedures are inspired by the guidelines in [11, 13].

Remark: Considering *also* the malware-specific classifiers as a stand-alone ML-NIDS is crucial: *(i)* an organization may use ML only to detect a specific malicious class; and *(ii)* it allows one to ascertain how well a "different" classifier can help compensate the deficiencies of another classifier when integrated in an ensemble.

[10] **Hardware:** We run our experiments on an Intel Xeon W-2195@2.3GHz(36 cores), 256GB RAM. Overall, our analysis took ≈3 weeks of computational runtime.

[11] We acknowledge that such a procedure (done also by [49]) may bear limitations. We favor consistency (instead of mixing data from different networks [11]). Nevertheless, our focus is on *malicious* data—which is not affected by this limitation.

Table 1: Pre-deployment results. We assess the performance of our ML-NIDS on the test set from "past" data. We report the average *tpr* (on malicious samples) and *tnr* (on benign samples). Cells in boldface are more relevant (they represent architectures denoting "generic" detectors). The defense is denoted with a 🛡.

		SCS: Aug.10th→Aug.18th, 2011					LCS: Feb.2017→Jul.2021				
		Full	**Ens**	Neris	Rbot	Virut	**Full**	**Ens**	Artemis	Dridex	Trickbot	Trickster	Wannacry
Benign	RF	**0.999**	**0.999**	0.999	1.000	1.000	**0.999**	**0.999**	0.999	0.999	0.999	0.999	0.999
	HGB	**0.999**	**0.999**	0.999	0.999	0.999	**0.999**	**0.999**	0.999	0.999	0.999	0.999	0.999
Malicious	RF	**0.994**	**0.992**	0.993	0.998	0.966	**0.999**	**0.999**	0.999	0.999	0.999	0.999	0.999
	HGB	**0.992**	**0.982**	0.995	0.999	0.763	**0.999**	**0.999**	0.999	0.999	0.999	0.999	0.999
Benign	🛡RF	**0.999**	**0.999**	0.999	0.999	0.999	**0.999**	**0.999**	0.999	0.999	0.999	0.999	0.999
	🛡HGB	**0.999**	**0.999**	0.999	0.999	0.999	**0.999**	**0.999**	0.999	0.999	0.999	0.999	0.999
Malicious	🛡RF	**0.993**	**0.992**	0.992	0.998	0.953	**0.999**	**0.999**	0.999	0.999	0.999	0.999	0.999
	🛡HGB	**0.989**	**0.983**	0.992	0.998	0.762	**0.999**	**0.999**	0.999	0.999	0.999	0.999	0.999

Table 2: Concept-drift results. We assess the performance of our ML-NIDS on the test set from "future" data. We report the *tpr* (malicious) and *tnr* (benign) averaged over 50 trials. Cells in red report cases in which the performance is statistically significantly ($p < 0.05$) *worse* than on the test set of "past" data (in Table 1). The defense is denoted with a 🛡.

		SCS: Aug.10th→Aug.18th, 2011					LCS: Feb.2017→Jul.2021				
		Full	**Ens**	Neris	Rbot	Virut	**Full**	**Ens**	Artemis	Dridex	Trickbot	Trickster	Wannacry
Benign	RF	**0.989**	**0.993**	0.993	1.000	1.000	**0.969**	**0.986**	0.999	0.993	0.991	0.999	0.998
	HGB	**0.990**	**0.982**	0.989	0.999	0.990	**0.959**	**0.965**	0.999	0.981	0.983	0.993	0.996
Malicious	RF	**0.675**	**0.587**	0.701	0.028	0.691	**0.927**	**0.988**	0.000	0.982	0.956	0.031	0.994
	HGB	**0.673**	**0.757**	0.768	0.020	0.663	**0.859**	**0.991**	0.010	0.977	0.970	0.031	0.995
Benign	🛡RF	**0.990**	**0.996**	0.996	0.999	0.999	**0.955**	**0.951**	0.995	0.965	0.986	0.992	0.997
	🛡HGB	**0.985**	**0.991**	0.995	0.998	0.995	**0.955**	**0.946**	0.995	0.962	0.984	0.992	0.996
Malicious	🛡RF	**0.631**	**0.438**	0.182	0.024	0.769	**0.786**	**0.957**	0.121	0.947	0.920	0.045	0.968
	🛡HGB	**0.634**	**0.561**	0.192	0.025	0.665	**0.791**	**0.958**	0.130	0.944	0.935	0.093	0.949

5.1 Verification Experiment (is our testbed valid?)

As a necessary step, we first ascertain whether our testbed meets the assumptions of our envisioned scenario (§2.2).

Pre-deployment (are our ML-NIDS good?) We train our ML-NIDS and assess their performance on the test set (i.e., from the "past" data), measuring the *tpr* and *tnr*: if these values are poor, then no organization would deploy such ML-NIDS to protect their networks. We report the results of this preliminary analysis in Table 1; boldface denotes the most important ML-NIDS (the "full" binary and the "ensemble"). We appreciate that the performance of all ML-NIDS is extremely high (aligning with prior work considering similar ML-NIDS [11]); interestingly, the Virut-specific HGB has a subpar *tpr*=0.763, but the corresponding ensemble has a much better one (0.982) thanks to the assistance of the other malware-specific classifiers. We also confirm that the same near-perfect performance is achieved by the corresponding "hardened" variants of our ML-NIDS (denoted with a 🛡): this is important for validation purposes, since the results claimed in the original publication of the defense [7] showed that, in the absence of adversarial perturbations (and concept drift), the defense did not cause significant performance degradation. In our supplementary document [3]), we report also the standard deviation (computed over the 50 trials).

Post-deployment (is there concept drift? We test our ML-NIDS on NetFlows from "future" data and report the results in Table 2. We see that there is a substantial drop in the *tpr* for all ML-NIDS in SCS, and a milder drop for those in LCS (albeit two malware-specific detectors can recognize almost no malicious NetFlows). The *tnr* also is lower, yielding more false positives. A similar degradation also affects the defense (🛡). We carry out t-tests: cells in red in Table 2 denote those cases in which the performance is statistically significantly *worse* ($p \ll 0.05$) than the baseline (from Table 1); our supplementary document [3] includes the standard deviations (computed over the 50 trials) and the description of the t-test. It is apparent that *our ML-NIDS are affected by concept drift*, as shown by the overwhelming number of red cells. In particular, the effects of such drift lead to *(i)* an increased number of false alarms—even a 0.01 lower *tnr* leads to thousands of benign NetFlows being misclassified; and to *(ii)* a significant number of false negatives—some detectors become completely unreliable, with *tpr* below 0.5

Takeaways. We derive two statistically significant ($p < 0.05$) results. **(1)** Our ML-NIDS exhibit high *tpr* and *tnr* in the pre-deployment phase—making them appropriate for our analysis. **(2)** Given our considered time-window, our ML-NIDS are affected by concept drift—validating our future experiments, and proving that `MCFP` *can be used for concept-drift assessments* by future work.

5.2 Attack Evaluation (do "blind" perturbations work?)

Given our prior results, we now focus on investigating our main objective, i.e., ascertaining the impact of "blind" adversarial perturbations under concept drift. Recall that our perturbations entail changes in a select subset of data (see §4.1). Hence, for a fair comparison, in this assessment we will only consider *(i)* UDP and TCP NetFlows which *(ii)* originate from the host "controlled" by the attacker. This latter requirement is crucial: first, to align with our threat model (realistically, "malicious" communications originating from *outside* a network are more likely to be blocked); second, to exclude "fictitious" adversarial NetFlows (some NetFlows within a "malicious" trace may entail communications that have nothing to do with the attacker's controlled host). This filtering allows us to consider only NetFlows denoting a physically realizable attack.

Non-adversarial results. To establish a baseline, we consider the *tpr* on our non-adversarial (but still malicious!) NetFlows from "future" data. We report the results in Table 3 (we stress that, due

Table 3: Non-adversarial results. We measure the average *tpr* on the "future" malicious NetFlows. We only consider UDP/TCP NetFlows starting from within the network. The defense is denoted with a ⛉: a ↑/↓ denotes when the defense is statistically significantly ($p < 0.05$) better/worse than the "vanilla".

		SCS: Aug.10th→Aug.18th, 2011					LCS: Feb.2017→Jul.2021						
		Full	**Ens**	Neris	Rbot	Virut	**Full**	**Ens**	Artemis	Dridex	Trickbot	Trickster	Wannacry
UDP	RF	**0.980**	**0.791**	0.941	0.151	0.384	**0.823**	**0.807**	0.000	0.785	0.832	0.000	0.166
	HGB	**0.824**	**0.816**	0.990	0.000	0.039	**0.754**	**0.893**	0.000	0.807	0.823	0.000	0.593
TCP	RF	**0.923**	**0.715**	0.366	0.436	0.773	**0.919**	**0.988**	0.000	0.937	0.922	0.996	0.997
	HGB	**0.943**	**0.861**	0.458	0.615	0.861	**0.983**	**0.993**	0.011	0.926	0.975	0.999	0.998
UDP	⛉RF	**0.892↓**	**0.304↓**	0.031↓	0.117	0.000↓	**0.698↓**	**0.908↑**	0.000	0.893↑	0.854↑	0.040↑	0.586↑
	⛉HGB	**0.921↑**	**0.478↓**	0.025↓	0.103↑	0.113↑	**0.705↓**	**0.909**	0.000	0.864↑	0.852↑	0.060↑	0.583↓
TCP	⛉RF	**0.880↓**	**0.814↑**	0.412↑	0.513↑	0.959↑	**0.979↑**	**0.958↓**	0.128↑	0.741↓	0.940	0.994	0.973↓
	⛉HGB	**0.867↓**	**0.810↓**	0.378↓	0.507↓	0.858	**0.982**	**0.976↓**	0.134↑	0.744↓	0.952↓	0.994↓	0.945↓

Table 4: Adversarial results. We compute the average *tpr* on "future" adversarial NetFlows. Cells in red denote cases in which the *tpr* is statistically significantly *worse* than the baseline in Table 3. A ↑/↓ denotes when the defense ⛉ is statistically significantly ($p < 0.05$) better/worse than the "vanilla".

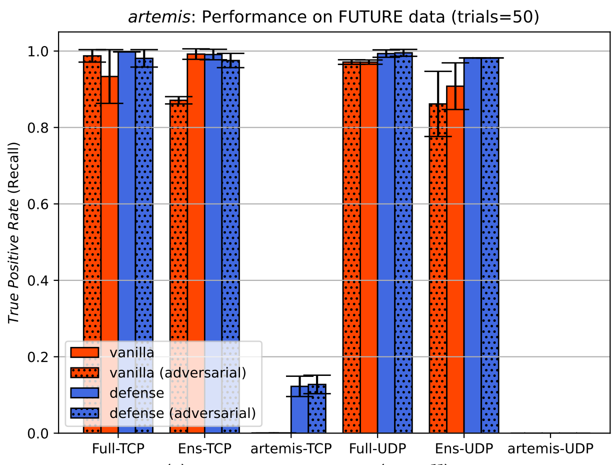
		SCS: Aug.10th→Aug.18th, 2011					LCS: Feb.2017→Jul.2021						
		Full	**Ens**	Neris	Rbot	Virut	**Full**	**Ens**	Artemis	Dridex	Trickbot	Trickster	Wannacry
UDP	RF	🟥 **0.921**	🟥 **0.602**	🟥 0.470	🟥 0.011	🟥 0.032	**0.812**	🟥 **0.700**	0.000	🟥 0.009	🟥 0.587	0.000	🟥 0.128
	HGB	**0.824**	**0.803**	0.975	0.014	0.059	**0.740**	**0.899**	0.000	🟥 0.660	0.833	0.000	0.593
TCP	RF	**0.902**	**0.717**	0.381	🟥 0.292	0.769	**0.929**	🟥 **0.934**	0.000	🟥 0.869	🟥 0.778	🟥 0.841	🟥 0.991
	HGB	**0.934**	**0.861**	0.472	🟥 0.470	0.822	🟥 **0.967**	🟥 **0.980**	0.022	🟥 0.910	🟥 0.856	🟥 0.298	🟥 0.992
UDP	⛉RF	**0.873↓**	**0.334↓**	0.052↓	🟥 0.050↑	0.000↓	**0.698↓**	**0.908↑**	0.000	0.881↑	0.855↑	🟥 0.020↑	0.586↑
	⛉HGB	🟥 **0.889↑**	🟥 **0.389↓**	0.060↓	0.079↑	0.058	**0.707**	**0.909**	0.000	0.826↑	0.856↑	0.080↑	0.581
TCP	⛉RF	**0.863↓**	🟥 **0.831↑**	0.400	🟥 0.378↑	0.953↑	🟥 **0.922**	**0.963↑**	0.127↑	🟥 0.721↓	🟥 0.742↓	🟥 0.499↓	0.969↓
	⛉HGB	**0.868↓**	**0.834**	0.414↓	🟥 0.410↓	0.816	🟥 **0.931↓**	**0.974↓**	0.132↑	🟥 0.651↓	🟥 0.762↓	🟥 0.507↑	0.949↓

to our filtering, these numbers are different from those in Table 2); Table 3 also includes the results of the hardened variants of our ML-NIDS (denoted with ⛉). Notably, on SCS, the defense is poor if applied to the ensemble for (non-adversarial) malicious UDP NetFlows (tpr=0.304 and 0.478 vs 0.791 and 0.816 for the vanilla ensemble); this result was not apparent in the original publication of the defense [7], which *did not account for concept drift.* However, the defense can help in LCS. We carry out a t-test to compare the *tpr* of the defense w.r.t. the "vanilla" ML-NIDS: a ↓ or ↑ in Table 3 denotes cases where the defense yields a statistically significantly ($p \ll 0.05$) inferior (18 cases) or superior (17 cases) *tpr* (for 13 cases, the defense does not lead to any change that is statistically significant).

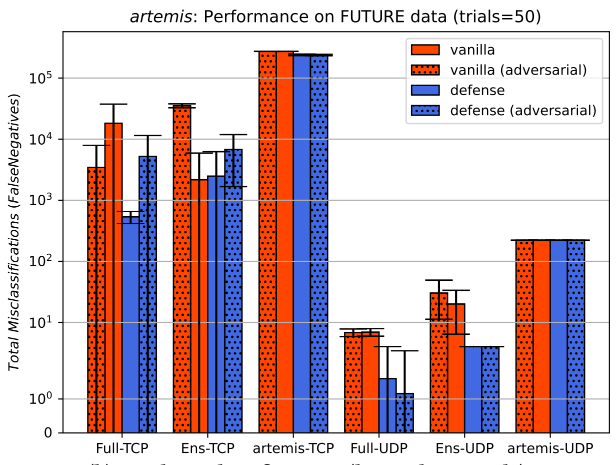
Adversarial results. We test our ML-NIDS on the respective adversarially-perturbed malicious NetFlows. First, we noticed that (as expected) there is no statistically significant difference between the "primary" and "secondary" variants of our adversarial traces (see §4.1). Hence, we report in Table 4 the *tpr* achieved by our ML-NIDS against the "primary" adversarial traces, showing the perturbations on the UDP and TCP perturbations (we report the results on the "secondary" traces in our supplementary document); the defense is also shown (⛉). Overall, we can see that our perturbations *do further degrade the tpr* w.r.t. Table 3. We carry out a t-test, comparing the results from Table 3 with those in Table 4, and report cases wherein the *tpr* is statistically significantly lower ($p \ll 0.05$) in red: worryingly, the RF is always affected by UDP perturbations on SCS, and both RF and HGB are almost always affected by TCP perturbations on LCS. From a *game-theory viewpoint*, the attacker should "use" UDP perturbations if he/she expects the ML-NIDS to rely on RF. Considering the defense, we note a counter-intuitive result in Table 4: even though the defense exhibits less cases with a statistically significant degradation than the vanilla (15 red cells for the defense vs 23 for the vanilla), *there are 20 cases (↓) where the defense is detrimental* to the *tpr*, and only 18 cases (↑) in which the defense is beneficial (w.r.t. the vanilla). This finding shows that drawing conclusions on the effectiveness of a defense based solely on whether the corresponding hardened model is more/less affected by adversarial perturbations can be misleading (echoing [76]).

We perform additional t-tests (described in our supplementary document [3]) to derive more statistically-sound conclusions. We confirm that our perturbations lead to a statistically significant ($p \ll 0.05$) decrease for all "vanilla" ML-NIDS in SCS, and all but three ML-NIDS in LCS: the Artemis- and Wannacry-specific classifiers (whose *tpr* is extremely low irrespective of the perturbations); and, remarkably, the Full-binary classifier (p=0.4>0.05): apparently, the effects of our perturbations on this classifier (for LCS) are not statistically significant. Then, we also confirm that the "defense" has underwhelming results (in some cases, it is *worse* than the vanilla). This latter finding contrasts with the conclusions of [7], based on feature-space perturbations (on a testbed similar to our SCS), further demonstrating that our paper underscores an "open problem."

Finally, we focus on some individual results. Consider the Ensemble classifier using HGB on LCS against UDP perturbations: tpr=0.899 vs 0.893 of the baseline (cf. Table 4 with Table 3), i.e., the perturbations *increase* the *tpr*. Interestingly, the Full-binary classifier has the opposite behaviour in the same setting (0.740 adversarial vs 0.754 non-adversarial). Despite these differences being not-statistically significant (see Table 4), we use these as a scaffold to investigate some intriguing phenomena: there may be some cases in which our perturbations are truly *detrimental for the attacker.* We stress that our perturbations are "blind", and hence not necessarily result in an "adversarial example" which is guaranteed to evade the targeted detector. Hence, this counterintuitive result is not to be taken as a flaw in our evaluation (see [54]).



(a) True-Positive Rate (Recall)



(b) Total Misclassifications (logarithmic scale)

Fig. 4: In-depth analysis of the results against the malicious NetFlows of Artemis (RF algorithm). It is important to note that the results of the Full-binary and of the Ensemble classifiers here (which consider *only* malicious NetFlows of Artemis) differ from those covered in §5.2 (which consider all malicious NetFlows).

Takeaways. (1) Our "blind" perturbations do cause a (mild, but statistically significant) performance degradation, but they have no effect on the Full-binary classifier on LCS (p=0.4). **(2)** The defense has a smaller benefit than reported in [7]. **(3)** In some isolated cases, our perturbations have little effect, and concept drift is enough to defeat the ML-NIDS.

5.3 Low-level Analysis (phenomena and explanations)

We conclude our assessment by elucidating (and trying to interpret) some findings that can only be appreciated with an in-depth look.

Use-case. We report in Figs. 4 the results on the NetFlows of Artemis (from LCS) achieved by the ML-NIDS using RF. Specifically, Fig.4a reports the *tpr*, whereas Fig. 4b reports the *total misclassifications*: such a twofold perspective (an absolute number and a percentage) allows one to appreciate what is happening at the micro level. We consider results on Artemis because it consistently yielded near-zero *tpr* by the Artemis-specific classifier (§5.2).

Analysis. First, we see that the defense always helps in this case (the *tpr* are always superior). Then, we see that the Full-binary classifier is *better* against our adversarial perturbations on TCP packets: from Fig. 4b, there are 18k misclassifications for non-adversarial NetFlows, and only 3.5k for adversarial NetFlows. The situation is inverted for the ensemble classifier, with 35k evasions due to adversarial NetFlows against only 2k for non-adversarial ones (making our perturbations very effective!). *Both of these claims are validated with a t-test* ($p \ll 0.05$). Then, we see that UDP (malicious) NetFlows are *always* misclassified by the Artemis-specific classifier: interestingly, the ensemble (which relies also on this classifier) still retains at least 0.871 *tpr*. This phenomenon is due to the 4 other malware-specific classifiers (i.e., Dridex, Trickster, Trickbot, Wannacry—all of which have *never seen* any NetFlow from Artemis during training!), of which at least one correctly predicted the ground truth of these NetFlows in the ensemble, leading to a final "malicious" classification.

Interpretation. We attempt to explain these intriguing results by focusing on the *size of the data* used to train and test our ML-NIDS [57]. By observing Table 6b in light of our cutoff date (1 July 2017), only the first trace of Artemis (captured on June 24, 2017) occurred in the "past", whereas all other occur in the "future". Hence, our ML-NIDS had only seen 70% of the 23k NetFlows of Artemis before being deployed (which still enabled to achieve near-perfect *tpr* on the remaining 30% of the "past" dataset—see Table 1). However, during "future", our ML-NIDS received 275k NetFlows from Artemis, whose variability (due to concept drift) bypassed the Artemis-specific classifier. In contrast, the full-binary and ensemble classifiers are trained also on other malicious classes, which boosted the baseline *tpr*. However, these classifiers exhibit a different behaviour against our perturbations on the NetFlows from Artemis: the ensemble was more fooled than the full-binary classifier. This is because, overall, the malware-specific classifiers of the ensemble had issues in classifying such NetFlows as malicious: their "fine-grained" perspective (each classifier is "aware" only of one type of malware) did not allow any of such classifiers to be sufficiently confident of the maliciousness of our adversarial NetFlows. In contrast, the more "generic" perspective of the Full-binary classifier (which is aware of all types of malware) induced this detector to consider some of our adversarially manipulated datapoints as malicious.

6 Discussion and Future Work

We now scrutinize our own contributions, drawing implications for related literature, and identifying avenues for future work.

Comparisons. As we discussed (§2 and §3.1), we are not aware of any prior work that considered the same assumptions made in our evaluation. Some papers (e.g. [9]) considered blind perturbations on CTU13 but in the feature space; Niu et al. [49] considered a mixed testbed (including also MCFP) but did not apply any perturbation and do not consider a time-aware evaluation. Although [72] try to assess the impact of concept drift (in a testbed entailing captures from vastly different networks and short timeframes) while also envisioning a black-box adversary, the actual attack requires query-access to the ML-NIDS, and the perturbations are applied in the feature space—making them not very realistic [8, 53]. Hence, due to the different underlying assumptions (i.e., testbed, threat model), we *cannot* compare our results with prior work: such comparisons would be unfair. Nevertheless, it is factual that the results we obtained (which are statistically validated) portray a substantially different reality than what was concluded in prior work.

Goal (and Extensions). The primary goal of our "open-problem" paper is to fairly investigate what happens when blind adversarial perturbations are used to evade an ML-NIDS under the impact of concept drift—which is a realistic setting for ML-NIDS that has not been explored before (§2.1). We are not interested in *(a)* "outperforming" prior work, or *(b)* "breaking" existing systems. We leave the development of a "solution" to this problem to *future work*—which is facilitated by our open-source code and detailed results [3]. There are virtually infinite ways to carry out an exploratory analysis to reach our goal. For instance, we could have considered another cutoff date; or different malicious classes; or more types of ML algorithms;[12] or feature sets/NetFlow exporters (e.g., Zeek [49]). We could also have: integrated some mechanism to mitigate concept drift (e.g., [5, 57, 75]—but doing so would violate our threat model!); or applied different adversarial manipulations; or used different defenses (e.g., adversarial training [6]). Our fully-documented and open-source tools [3], nonetheless, allow *future research* to investigate all of these additional contexts. Indeed, reproducing such contexts is trivial with our tools: our packet modifier (§4.1) can be used to craft different problem-space perturbations; the resulting PCAP trace can then be processed with different NetFlow exporters, leading to different feature representations. Moreover, changing the cutoff-date is a one-liner in our code: such a change can allow one to investigate the performance of the resulting model over longer or shorter timespans.[13] Finally, the "open problem" tackled in our paper should serve as an inspiration to *(i)* study the effects of previously proposed mechanisms to counter concept drift by accounting for longer timespans (e.g., [5, 75] consider only one week); or *(ii)* attempt to further explain our results (as done in [33, 47]).

Lessons Learned. We derive three relevant implications for future endeavours. **(1)** The MCFP dataset and our custom resources [3] can be used by future research for realistic concept-drift and/or problem-space assessments of adversarial perturbations in ML-NIDS contexts. **(2)** Overall, blind perturbations (when applied on raw network traffic and in the presence of concept drift) can decrease the tpr of state-of-the-art ML-NIDS. However, some perturbations have no effect, or can be detrimental to the attacker. We endorse future work to consider *game-theory* approaches [64]: by modeling this scenario as a "win/lose" game, it may be possible to find the optimal solution (i.e., the one that leads to higher chances of winning for any given "player"). **(3)** Statistical tests are pivotal to make sound claims. For instance, in some cases our perturbations lowered the tpr, but the impact was not statistically significant (i.e., $p > 0.05$). Yet, we are not aware of any prior work on concept drift in the NIDS context whose claims were validated via statistical tests. We suggest future work to repeat their experiments multiple times and derive statistically sound conclusions.[14]

[12] We did experiments also on **deep-learning** classifiers: they required 100x the training time and yielded inferior performance than RF and HGB—so we excluded these.

[13] Our exploratory analysis covers a variety of use cases from a temporal perspective. From Table 6, we see that the "future" datapoints of some pieces of malware (e.g., Dridex, Trickbot, Trickster) occur more than 6 months after the model was trained; whereas for others (e.g., Wannacry, Artemis) this temporal window is much shorter.

[14] We show how to carry out these verifications in a 40s video (included in our repo [3]).

7 Conclusions

We carried out a realistic assessment of state-of-the-art ML techniques for NIDS when they are simultaneously subject to concept drift and "blind" adversarial ML attacks. Our envisioned attacker is more constrained than the one of most prior work, but our adversarial perturbations still degrade the performance of the ML-NIDS. Plus, a defense deemed practical against perturbations in the feature space is underwhelming in our considered scenario.

Altogether, our paper highlights that research on concept drift in ML-NIDS contexts is a much more "open problem" than what may have appeared: our contributions serve as a stepping stone to reassess these intriguing phenomena, which are of pivotal importance for real-world deployments of ML in NIDS.

Acknowledgement

We thank the reviewers for the feedback. This research was funded by the EPSRC Grant EP/X015971/1 and by Hilti.

References

[1] 2021. MCFP. https://www.stratosphereips.org/datasets-malware.
[2] 2024. Class B Network. service.snom.com/display/wiki/IP+address+classes.
[3] 2024. Our Repository. https://github.com/hihey54/aisec24.
[4] Bushra A Alahmadi, Louise Axon, and Ivan Martinovic. 2022. 99% False Positives: A Qualitative Study of SOC Analysts' Perspectives on Security Alarms. In *USENIX Security*.
[5] Giuseppina Andresini, Feargus Pendlebury, Fabio Pierazzi, Corrado Loglisci, Annalisa Appice, and Lorenzo Cavallaro. 2021. INSOMNIA: Towards Concept-drift Robustness in Network Intrusion Detection. In *ACM AISec*.
[6] Giovanni Apruzzese, Hyrum Anderson, Savino Dambra, David Freeman, Fabio Pierazzi, and Kevin Roundy. 2023. "Real Attackers Don't Compute Gradients": Bridging the Gap Between Adversarial ML Research and Practice. In *IEEE SaTML*.
[7] Giovanni Apruzzese, Mauro Andreolini, Michele Colajanni, and Mirco Marchetti. 2020. Hardening random forest cyber detectors against adversarial attacks. *IEEE TETCI* (2020).
[8] Giovanni Apruzzese, Mauro Andreolini, Luca Ferretti, Mirco Marchetti, and Michele Colajanni. 2021. Modeling Realistic Adversarial Attacks against Network Intrusion Detection Systems. *ACM DTRAP* (2021).
[9] Giovanni Apruzzese and Michele Colajanni. 2018. Evading botnet detectors based on flows and Random Forest with adversarial samples. In *IEEE NCA*.
[10] Giovanni Apruzzese, Pavel Laskov, Edgardo Montes de Oca, Wissam Mallouli, Luis Búrdalo Rapa, Athanasios Vasileios Grammatopoulos, and Fabio Di Franco. 2022. The Role of Machine Learning in Cybersecurity. *ACM DTRAP* (2022).
[11] Giovanni Apruzzese, Pavel Laskov, and Johannes Schneider. 2023. SoK: Pragmatic Assessment of Machine Learning for Network Intrusion Detection. In *EuroS&P*.
[12] Giovanni Apruzzese, Aliya Tastemirova, and Pavel Laskov. 2022. SoK: The Impact of Unlabelled Data in Cyberthreat Detection. In *IEEE EuroS&P*.
[13] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *USENIX Security*.
[14] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2022. Transcending transcend: Revisiting malware classification in the presence of concept drift. In *IEEE S&P*.
[15] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Elsevier Pattern Recogn.* (2018).
[16] Branislav Bosansky, Lada Hospodkova, Michal Najman, Maria Rigaki, Elnaz Babayeva, and Viliam Lisy. 2024. Counteracting Concept Drift by Learning with Future Malware Predictions. *arXiv:2404.09352* (2024).
[17] Emilie Bout, Valeria Loscri, and Antoine Gallais. 2021. How machine learning changes the nature of cyberattacks on IoT networks: A survey. *IEEE COMST* (2021).
[18] Tobias Braun, Irdin Pekaric, and Giovanni Apruzzese. 2024. Understanding the Process of Data Labeling in Cybersecurity. In *SAC*.
[19] Marta Catillo, Antonio Pecchia, and Umberto Villano. 2023. Machine Learning on Public Intrusion Datasets: Academic Hype or Concrete Advances in NIDS?. In *IEEE DSN*.
[20] Romain Cayre, Vincent Nicomette, Guillaume Auriol, Mohamed Kaâniche, and Aurélien Francillon. 2024. OASIS: An Intrusion Detection System Embedded in Bluetooth Low Energy Controllers. In *ACM AsiaCCS*.
[21] Antonio Emanuele Cinà, Kathrin Grosse, Ambra Demontis, Sebastiano Vascon, Werner Zellinger, Bernhard A Moser, Alina Oprea, Battista Biggio, Marcello

Pelillo, and Fabio Roli. 2023. Wild patterns reloaded: A survey of machine learning security against training data poisoning. *ACM CSUR* (2023).

[22] Henry Clausen, Gudmund Grov, and David Aspinall. 2021. Cbam: A contextual model for network anomaly detection. *Computers* (2021).

[23] Igino Corona, Giorgio Giacinto, and Fabio Roli. 2013. Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues. *Inf. Sci.* (2013).

[24] Luca Demetrio, Scott E Coull, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. 2021. Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection. *ACM TOPS* (2021).

[25] Marius Dragoi, Elena Burceanu, Emanuela Haller, Andrei Manolache, and Florin Brad. 2022. AnoShift: A distribution shift benchmark for unsupervised anomaly detection. *NeurIPS* (2022).

[26] Juliette Dromard and Philippe Owezarski. 2020. Study and evaluation of unsupervised algorithms used in network anomaly detection. In *FTC*.

[27] Gints Engelen, Vera Rimmer, and Wouter Joosen. 2021. Troubleshooting an intrusion detection dataset: the CICIDS2017 case study. In *IEEE S&PW*.

[28] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. 2014. An empirical comparison of botnet detection methods. *Comput. Secur.* (2014).

[29] Dongqi Han, Zhiliang Wang, Wenqi Chen, Kai Wang, Rui Yu, Su Wang, Han Zhang, Zhihua Wang, Minghui Jin, Jiahai Yang, et al. 2023. Anomaly Detection in the Open World: Normality Shift Detection, Explanation, and Adaptation. In *NDSS*.

[30] Dongqi Han, Zhiliang Wang, Ying Zhong, Wenqi Chen, Jiahai Yang, Shuqiang Lu, Xingang Shi, and Xia Yin. 2021. Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors. *IEEE J. Selected Areas in Communications* (2021).

[31] Ke He, Dan Dongseong Kim, and Muhammad Rizwan Asghar. 2023. Adversarial machine learning for network intrusion detection systems: a comprehensive survey. *IEEE COMST* (2023).

[32] Thomas Hutzelmann, Dominik Mauksch, Ana Petrovska, and Alexander Pretschner. 2023. Generation of Tailored and Confined Datasets for IDS Evaluation in Cyber-Physical Systems. *IEEE TDSC* (2023).

[33] Arthur S Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A Ferreira, Arpit Gupta, and Lisandro Z Granville. 2022. AI/ML for Network Security: The Emperor has no Clothes. In *ACM CCS*.

[34] Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Arjun Nitin Bhagoji, Paul Schmitt, Francesco Bronzino, and Nick Feamster. 2024. NetDiffusion: Network Data Augmentation Through Protocol-Constrained Traffic Generation. *ACM MACS* (2024).

[35] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting concept drift in malware classification models. In *USENIX Security*.

[36] Zilong Lin, Yong Shi, and Zhi Xue. 2022. Idsgan: Generative adversarial networks for attack generation against intrusion detection. In *PAKDD*.

[37] Lisa Liu, Gints Engelen, Timothy Lynar, Daryl Essam, and Wouter Joosen. 2022. Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018. In *IEEE CNS*.

[38] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. 2018. Learning under concept drift: A review. *IEEE TKDE* (2018).

[39] Nuno Martins, José Magalhães Cruz, Tiago Cruz, and Pedro Henriques Abreu. 2020. Adversarial Machine Learning applied to Intrusion and Malware Scenarios: a systematic review. *IEEE Access* (2020).

[40] Vinicius Eiji Martins, Alberto Cano, and Sylvio Barbon Junior. 2023. Meta-learning for dynamic tuning of active learning on stream classification. *Pattern Recognition* (2023).

[41] Jovana Mijalkovic and Angelo Spognardi. 2022. Reducing the false negative rate in deep learning based network intrusion detection systems. *Algorithms* (2022).

[42] Brad Miller, Alex Kantchelian, Sadia Afroz, Rekha Bachwani, Edwin Dauber, Ling Huang, Michael Carl Tschantz, Anthony D Joseph, and J Doug Tygar. 2014. Adversarial active learning. In *ACM AISec*.

[43] Céline Minh, Kevin Vermeulen, Cédric Lefebvre, Philippe Owezarski, and William Ritchie. 2023. An explainable-by-design ensemble learning system to detect unknown network attacks. In *CNSM*.

[44] Jaron Mink, Hadjer Benkraouda, Limin Yang, Arridhana Ciptadi, Ali Ahmadzadeh, Daniel Votipka, and Gang Wang. 2023. Everybody's Got ML, Tell Me What Else You Have: Practitioners' Perception of ML-Based Security Tools and Explanations. In *IEEE S&P*.

[45] Omid Mirzaei, Guillermo Suarez-Tangil, Jose M de Fuentes, Juan Tapiador, and Gianluca Stringhini. 2019. Andrensemble: Leveraging api ensembles to characterize android malware families. In *ACM AsiaCCS*.

[46] Deepa Mulimani, Shashikumar G Totad, Prakashgoud Patil, and Shivananda V Seeri. 2021. Adaptive ensemble learning with concept drift detection for intrusion detection. In *ICICC*.

[47] Azqa Nadeem, Daniël Vos, Clinton Cao, Luca Pajola, Simon Dieck, Robert Baumgartner, and Sicco Verwer. 2023. Sok: Explainable machine learning for computer security applications. In *IEEE EuroS&P*.

[48] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2021. Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations. In *USENIX Security*.

[49] Zequn Niu, Jingfeng Xue, Dacheng Qu, Yong Wang, Jun Zheng, and Hongfei Zhu. 2022. A novel approach based on adaptive online analysis of encrypted traffic for identifying Malware in IIoT. *Inf. Sci.* (2022).

[50] Philippe Owezarski. 2023. Investigating adversarial attacks against Random Forest-based network attack detection systems. In *IEEE NOMS*.

[51] Marc-Oliver Pahl and François-Xavier Aubet. 2018. All eyes on you: Distributed Multi-Dimensional IoT microservice anomaly detection. In *IEEE CNSM*.

[52] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *USENIX Security*.

[53] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. 2020. Intriguing Properties of Adversarial ML Attacks in the Problem Space. In *IEEE S&P*.

[54] Maura Pintor, Luca Demetrio, Angelo Sotgiu, Ambra Demontis, Nicholas Carlini, Battista Biggio, and Fabio Roli. 2022. Indicators of attack failure: Debugging and improving optimization of adversarial examples. *NeurIPS* (2022).

[55] Michal Piskozub, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. 2019. On the resilience of network-based moving target defense techniques against host profiling attacks. *Wmtd* (2019).

[56] QoSient. 2012. Argus NetFlow. https://qosient.com/argus/argusnetflow.shtml.

[57] William K Robertson, Federico Maggi, Christopher Kruegel, Giovanni Vigna, et al. 2010. Effective Anomaly Detection with Scarce Training Data.. In *NDSS*.

[58] SANS. 2023. *2023 SOC Survey*. Technical Report.

[59] Giorgio Severi, Simona Boboila, Alina Oprea, John Holodnak, Kendra Kratkiewicz, and Jason Matterer. 2023. Poisoning Network Flow Classifiers. In *ACSAC*.

[60] Giorgio Severi, Jim Meyer, Scott Coull, and Alina Oprea. 2021. Explanation-Guided backdoor poisoning attacks against malware classifiers. In *USENIX Sec.*

[61] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *ICISSP*.

[62] Anshuman Singh, Andrew Walenstein, and Arun Lakhotia. 2012. Tracking concept drift in malware families. In *ACM AISec*.

[63] Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE S&P*.

[64] Guoxin Sun, Tansu Alpcan, Seyit Camtepe, Andrew C Cullen, and Benjamin IP Rubinstein. 2023. An Adversarial Strategic Game for Machine Learning as a Service using System Features.. In *AAMAS*.

[65] Fnu Suya, Anshuman Suri, Tingwei Zhang, Jingtao Hong, Yuan Tian, and David Evans. 2024. Sok: Pitfalls in evaluating black-box attacks. In *IEEE SaTML*.

[66] Thijs Van Ede, Hojjat Aghakhani, Noah Spahn, Riccardo Bortolameotti, Marco Cova, Andrea Continella, Maarten van Steen, Andreas Peter, Christopher Kruegel, and Giovanni Vigna. 2022. Deepcase: Semi-supervised contextual analysis of security events. In *IEEE S&P*.

[67] Mathew Vermeer, Michel Van Eeten, and Carlos Gañán. 2022. Ruling the rules: Quantifying the evolution of rulesets, alerts and incidents in network intrusion detection. In *ACM AsiaCCS*.

[68] Nikos Virvilis and Dimitris Gritzalis. 2013. The big four-what we did wrong in advanced persistent threat detection?. In *ARES*.

[69] Gernot Vormayr, Joachim Fabini, and Tanja Zseby. 2020. Why are my flows different? A tutorial on flow exporters. *IEEE COMST* (2020).

[70] Omar Abdel Wahab. 2022. Intrusion detection in the iot under data and concept drifts: Online deep learning approach. *IEEE IoT J.* (2022).

[71] Ning Wang, Yimin Chen, Yang Xiao, Yang Hu, Wenjing Lou, and Y Thomas Hou. 2022. Manda: On adversarial example detection for network intrusion detection system. *IEEE TDSC* (2022).

[72] Xian Wang. 2022. ENIDrift: A Fast and Adaptive Ensemble System for Network Intrusion Detection under Real-world Drift. In *ACSAC*.

[73] Di Wu, Binxing Fang, Junnan Wang, Qixu Liu, and Xiang Cui. 2019. Evading machine learning botnet detection models via deep reinforcement learning. In *IEEE ICC*.

[74] Anli Yan, Zhenxiang Chen, Riccardo Spolaor, Shuaishuai Tan, Chuan Zhao, Lizhi Peng, and Bo Yang. 2020. Network-based malware detection with a two-tier architecture for online incremental update. In *IWQoS*.

[75] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. 2021. CADE: Detecting and explaining concept drift samples for security applications. In *USENIX Security*.

[76] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. 2017. Efficient defenses against adversarial attacks. In *ACM AISec*.

[77] Chaoyun Zhang, Xavier Costa-Perez, and Paul Patras. 2022. Adversarial attacks against deep learning-based network intrusion detection systems and defense mechanisms. *TON* (2022).

[78] Ying Zhong, Yiran Zhu, Zhiliang Wang, Xia Yin, Xingang Shi, and Keqin Li. 2020. An adversarial learning model for intrusion detection in real complex network environments. *WASA* (2020).

Table 5: Benign PCAP traces from `MCFP`, containing "background" and "active" traffic.

Trace (Link)	Date	PCAP Size(B)	Flows Total
1	17 Dec 2013	400M	10K
2	17 Dec 2013	800M	10K
3	24 Mar 2015	1M	3K
4	13 Sept 2016	0.6M	40
5	13 Sept 2016	0.6M	50
6	13 Sept 2016	0.5M	40
7	18 Apr 2017	280M	18K
8	19 Apr 2017	200M	7.8K
9	25 Apr 2017	480M	7.8K
10	26 Apr 2017	110M	16K
11	30 Apr 2017	270M	21K
12	30 Apr 2017	424M	39K
13	1 May 2017	137M	12K
14	1 May 2017	435M	35K
15	1 May 2017	800M	55K
16	1 May 2017	725M	60K
17	2 May 2017	300M	15K
18	2 May 2017	1.6G	102K
19	2 May 2017	822M	82K
20	3 Jul 2017	0.5M	34
21	23 Jul 2017	400M	6.2K
22	5 Sept 2017	16M	1K
23	7 May 2018	300M	4.9K

(a) Recent benign traces (active).

Trace (Link)	Date	PCAP Size(B)	Flows Total	Nature
42	10 Aug 2011	6.1G	4M 35K	Background Active
43	11 Aug 2011	6.2G	2.5M 10K	Background Active
44	12 Aug 2011	14.5G	5M 113K	Background Active
45	15 Aug 2011	5.4G	1.4M 30K	Background Active
46	15 Aug 2011	0.4G	150K 5K	Background Active
54	16 Aug 2011	4.4G	2.1M 28K	Background Active
50	17 Aug 2011	9.8G	2.6M 35K	Background Active
52	18 Aug 2011	0.6G	113K 2.8K	Background Active

(b) Traces from `CTU13`.

Table 6: Malicious PCAP traces from `MCFP` used in our assessment (we will perturb these). Note: the most recent traces in `MCFP` are collected in 2021. We provide an explanation of the method used to choose these traces in our supplementary document [3].

m	Trace (Link)	Date	PCAP Size(B)	Flows Total
Wannacry	1	14 May 2017	500K	5K
	2	14 May 2017	11M	15K
	3	15 May 2017	3.6M	171
	4	15 May 2017	13M	32K
	5	24 Jun 2017	444M	9K
	6	11 Jul 2017	1.6M	14K
	7	11 Jul 2017	7.6M	14K
	8	11 Jul 2017	7.3M	13K
	9	11 Jul 2017	7.1M	11K
	10	11 Jul 2017	6.8M	9.3K
	11	11 Jul 2017	3.1M	35
	12	11 Jul 2017	6.3M	4K
	13	11 Jul 2017	14M	17K
	14	12 Jul 2017	6.1M	3.6K
	15	13 Jul 2017	6.2M	210
	16	13 Jul 2017	6.8M	11K
	17	13 Jul 2017	6.7M	10K
Dridex	1	13 Feb 2017	79M	102
	2	27 Feb 2017	57M	2.5K
	3	11 Apr 2017	31M	51K
	4	18 Apr 2017	66M	30K
	5	18 Apr 2017	47M	35K
	6	15 May 2017	7.4M	43K
	7	15 May 2017	33M	48K
	8	16 May 2017	52M	63K
	9	24 Jun 2017	16M	11K
	10	29 Jan 2018	310M	73K
	11	30 Jan 2018	193M	37K
	12	03 Apr 2018	223M	52K

(a) More recent traces (part 1).

m	Trace (Link)	Date	PCAP Size(B)	Flows Total
Artemis	1	24 Jun 2017	37M	23K
	2	1 Aug 2017	336M	30K
	3	14 Aug 2017	772M	226K
	4	16 Aug 2017	153M	11K
	5	16 Aug 2017	146M	10K
Trickster	1	24 Jun 2017	52M	24K
	2	3 Aug 2017	6.4M	2K
	3	29 Jan 2018	252M	63K
Trickbot	1	29 Mar 2017	83M	40K
	2	30 Mar 2017	90M	41K
	3	30 Mar 2017	90M	41K
	4	30 Mar 2017	81M	38K
	5	12 Apr 2017	288M	160K
	6	12 Apr 2017	115M	53K
	7	17 Apr 2017	142M	103K
	8	8 May 2017	214M	127K
	9	15 May 2017	204M	79K
	10	7 Jun 2017	211M	124K
	11	15 Jun 2017	228M	141K
	12	24 Jun 2017	77M	31K
	13	24 Jun 2017	76M	33K
	14	24 Jun 2017	78M	31K
	15	24 Jun 2017	44M	27K
	16	30 Jan 2018	33M	13K
	17	30 Jan 2018	212M	62K
	18	2 Feb 2018	197M	59K
	19	27 Mar 2018	410M	122K
	20	30 Jul 2021	100K	61

(b) More recent traces (part 2).

m	Trace (Link)	Date	PCAP Size(B)	Flows Total
Neris	1	10 Aug 2011	56M	42K
	2	11 Aug 2011	35M	23K
	3	17 Aug 2011	1G	188K
Rbot	1	12 Aug 2011	123M	40K
	2	15 Aug 2011	212M	0.8K
	3	18 Aug 2011	4G	8.8K
Virut	1	15 Aug 2011	30M	0.9K
	2	16 Aug 2011	110M	41K

(c) Traces from `CTU13`.

Appendix A What are NetFlows?

NetFlows are metadata that provide high-level statistics on a set of packets having: same source and destination hosts, same source and destination ports, and same protocol. Depending on the specific implementation, various information (i.e., the "features" that can be used for ML-based analyses) can be extracted in any given NetFlow (e.g., total number of packets or bytes exchanged, or duration of the communication). For more details, see [11, 69].