"What is the Problem Space?" Defining Host-space Adversarial Perturbations against Network Intrusion Detection Systems

Miel Verkerken*
Ghent University - imec
Ghent, Belgium
miel.verkerken@ugent.be

Laurens D'hooge
Ghent University - imec
Ghent, Belgium
laurens.dhooge@ugent.be

Bruno Volckaert
Ghent University - imec
Ghent, Belgium
bruno.volckaert@ugent.be

Filip De Turck
Ghent University - imec
Ghent, Belgium
filip.deturck@ugent.be

Giovanni Apruzzese
University of Liechtenstein
Vaduz, Liechtenstein
Reykjavik University
Reykjavik, Iceland
giovannia@ru.is

Abstract

Network Intrusion Detection Systems (NIDS) are now increasingly leveraging Machine Learning (ML) techniques to detect malicious network activities. Numerous papers have scrutinized the security of ML-based NIDS (ML-NIDS) by testing them against various attacks involving adversarial perturbations. The findings were oftentimes worrying: by making imperceptible changes to a given input, powerful ML models would be bypassed. In this context, we took a step back and wondered: where (i.e., in what "space") have these perturbations been applied?

We argue that real-world adversaries can apply adversarial perturbations only by operating on the hosts they can control—a concept which we define as *host-space perturbations*. To some, such an observation may seem trivial. And yet, through a systematic literature review (n=316), we found that prior work applied perturbations by manipulating pre-collected datapoints (e.g., a packet *captured by the router*, or a network flow *analysed by the ML-NIDS*). Such operations, while not impossible, may be outside the reach of an attacker who can only control some (unprivileged) hosts in a network. Hence, to demonstrate how to craft host-space perturbations and study some of their effects, we experimented on well-known benchmarks and a real-world network. We show that ML-NIDS that can detect the SSH-bruteforcing attempts launched via a given command string cannot detect any attempt launched by changing *a single character* of such a string. We then examined how such a minuscule change in the "problem space" (i.e., the attacker's host) can lead to devastating effects on the "feature space". We derive lessons learned on how to practically assess host-space perturbations. Our stance is that the security of ML-NIDS should be re-assessed.

*Please contact Miel Verkerken for inquiries also at mielverkerken@hotmail.com

ASIA CCS '26, Bangalore, India

ACM ISBN 979-8-4007-2356-8/26/06
https://doi.org/10.1145/3779208.3807482

CCS Concepts

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

Keywords

Evasion, Adversarial ML Attacks, Out Of Distribution

ACM Reference Format:
Miel Verkerken, Laurens D'hooge, Bruno Volckaert, Filip De Turck, and Giovanni Apruzzese. 2026. "What is the Problem Space?" Defining Host-space Adversarial Perturbations against Network Intrusion Detection Systems. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '26), June 1–5, 2026, Bangalore, India.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3779208.3807482

1 Introduction

Assessing the security of Machine Learning (ML) systems is challenging. On the one hand, researchers must foresee the various ways in which a hypothetical attacker can interact with the envisioned ML system [28]. On the other hand, such interactions must be faithfully reproduced in experiments that resemble a real-world setup [87]. Achieving both in the context of modern networks, known for their immense variability [106], is objectively hard.

Yet, in the specific domain of Network Intrusion Detection Systems (NIDS), there is a never-ending need of such assessments. Real-world evidence shows that ML models are no longer a "research toy". Rather, they are increasingly integrated in systems deployed in full-fledged security operation centers [36–39, 49, 77, 105, 113] that protect the networks of modern organizations. Despite hundreds of works studying the security and robustness of ML-driven NIDS (ML-NIDS) in adversarial environments (see, e.g., [15, 59, 120]), we assert that *the real-world fidelity of the experimental approaches adopted in related research still remains an open issue.*

Recent seminal works have highlighted that real attackers operate in the "problem space" [87] and must deal with complex ML systems [14], which can be outside the attacker's reach. And yet, numerous works persist in adopting an ML-centric view: for instance, subsequent works accepted to S&P'24 [44] and EuroS&P'24 [26] used well-known "gradient-based" attacks [68] to evade an ML-NIDS trained on the CICIDS17 benchmark dataset [100]. Such adversarial perturbations are applied to the feature representation of

a given input: despite substantially reducing the baseline accuracy (e.g., from 99% to 16%), these "adversarial examples" may not be physically realizable by a real-world attacker, who cannot violate strict domain constraints [101]. Some works introduced perturbations in "spaces" that are closer to the attacker, such as manipulating pre-collected network packets [18, 59]. Yet, even these perturbations are applied in the real problem space: to reliably change the packets analysed by an ML-NIDS, an attacker requires access to a highly privileged device (e.g., the router, or the NIDS itself [15]). Is this possible? Without a doubt. But is this what an attacker would, or can, do in practice? (This is an open question to the reader)

Contributions. We posit that real attackers seeking to evade an ML-NIDS would first attempt to do so by issuing slightly different commands on the host they can directly control—and not by manipulating datapoints collected by other devices. *We define such a tactic as a "host-space perturbation"* (§3). As we will show with a systematic literature review across 316 papers (§4), this specific class of adversarial perturbations has been overlooked by prior work. So, we took a first step (§5) and assessed what happens if an attacker, instead of launching `patator persistent=1` in an attempt to bruteforce an ssh server [5], uses `patator persistent=0` instead: an ML-NIDS having perfect detection rate against the former command cannot detect a single NetFlow of the latter (we tested this both on benchmark datasets and in a real-world network). We thoroughly examine this use case (§6), explaining the reasons behind these results, and also highlighting the pitfalls of replicating host-space perturbations by manipulating pre-collected data (i.e., the de facto way of applying adversarial perturbations). We also expand our assessment (in the Appendix C) by considering additional attack types, affected ML models, and network environments. Based on our findings, we therefore advocate for future work to revisit the approaches used for security assessments of ML-NIDS and provide recommendations on how to replicate host-space perturbations in a research setting (§8). We release all our experimental resources (for which we used open-source tools), including the data of our real-world evaluation, in our repository [6].

2 Background and Motivation

We summarize the field of ML-NIDS (§2.1) and of adversarial ML (§2.2) before making our problem statement (§2.3).

2.1 Primer on ML-driven NIDS

Research on Intrusion Detection has spanned almost forty years [43], and intrusion-detection systems (IDS) have been studied by thousands of publications [41]. IDS are designed to *detect intrusions*: in other words, they implicitly assume that an "intrusion" has taken place, and hence seek to warn users (e.g., by raising "alerts") as early as possible to contain the damage that an attacker may cause to the targeted system [70].

Over the years, IDS have been linked to a variety of terms covered in a plethora of taxonomies (e.g., [42, 114]) and surveys [11, 66, 71, 112]. Here, we consider a specific class of IDS: those using machine-learning (ML) techniques applied to network-related data. By "ML techniques", we intend automated data-driven methods whose detection capabilities are determined by a training phase, during which a given ML model (e.g., a decision tree, or a neural network) learns to make decisions by observing (typically "large") previously collected datasets. By "network-related data", we intend data pertaining to the communications occurring within a given network—such as, e.g., network traffic included in a packet-capture (PCAP) trace [55], or network flows (NetFlows) providing high-level summaries of the communications between two endpoints [25, 119]. Such endpoints not necessarily involve different and/or physical devices (e.g., they can entail virtual machines, or even communications sent from, and directed to, the same host). We therefore denote our considered class of IDS as ML-based Network-IDS, or ML-NIDS (note that our focus overlaps with that of a recent SoK [19]).

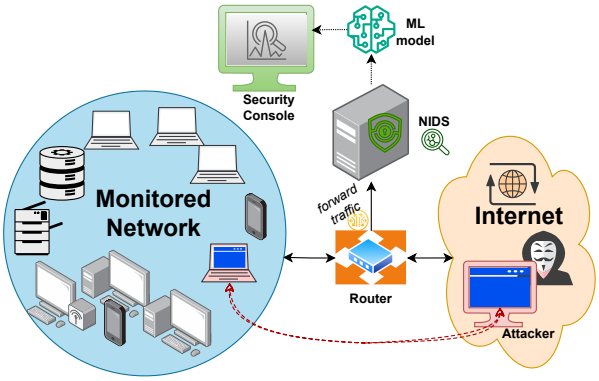


Fig. 1: Typical NIDS scenario. N.b.: the attacker *cannot* control the router or the NIDS (otherwise, it wouldn't be surprising if the NIDS is bypassed.

We provide a visualization of an exemplary ML-NIDS deployment scenario in Fig. 1. At a high level, the NIDS receives the network-related data pertaining to a monitored network. Specifically, such data is forwarded to the NIDS by a dedicated network appliance (e.g., a router). The NIDS then analyses such data by sending it to any given ML model—which can be seen as a classifier that "detects" if the data contains traces of an "intrusion" that must be escalated to a security operator, who must then take action. ML-NIDS have been widely studied in academic literature, and prior work showed that, e.g., ML-NIDS can detect malware [54, 88], botnet [56, 78], DDoS [9, 45], reconnaissance [57, 118], lateral-movement [23, 89], or bruteforcing [80, 84] attacks. Indeed, even in the real world, ML methods are commonly deployed in security-operation centers, or by world-renowned networking companies [36] to detect malicious activities [38].

2.2 Adversarial Perturbations vs ML Models

The growth of ML led to many studies scrutinising the security of these techniques when deployed in adversarial settings, giving rise to the field of "adversarial machine learning" [62].

Thousands of papers have highlighted that ML models are vulnerable to "adversarial perturbations", i.e., tiny manipulations that adversely affect an ML model's performance [28]. There are various ways to convey such "adversarial ML attacks" (e.g., [14, 27, 35, 86]). For instance, attackers can attempt to poison the ML model by tampering with its training phase [35]; or attempt to evade its detection during its inference phase [14]. Unfortunately, adversarial ML attacks can also target ML models designed for NIDS [15].

Assessment of adversarial ML attacks requires, from a research viewpoint, determining where (i.e., in which "space") the perturbation is applied [87]. Historically, adversarial perturbations had been

applied in the *feature space*, i.e., by directly changing the feature vector provided as input to the ML model.[1] However, as pointed out by Pierazzi et al. [87], *real attackers operate in the "problem" space*—which not necessarily overlaps with the feature space [17]. Indeed, as noted by more recent works [14], real-world attackers interact with ML systems—and not ML models.

Such an observation highlights a crucial issue: in a lab setting, manipulations in the feature space are meant to approximate the operations that an attacker would perform in the real world; however, careless manipulation of the feature space risks creating "adversarial examples" that may not be physically realizable and/or which violate domain constraints [101].[2]

2.3 Shortcomings of Adversarial Perturbations

It can be said that the paper by Pierazzi et al. [87] questioned the real-world validity of the results portrayed by previous research, since feature-space perturbations were the norm[3] in the adversarial ML domain—including the specific context of ML-NIDS [15].

As a consequence, some works (e.g., [59, 120]) began to explore scenarios wherein the perturbations were not applied to the input features, but in other spaces—allegedly being a better representation of a real attacker's operations. For instance, Han et al. [59] crafted "traffic-space perturbations" by manipulating a PCAP trace, adding bytes to the packets' payload: such changes would propagate to the feature space, thereby influencing the ML-NIDS; whereas Wang et al. [120] claimed to generate "problem-space adversarial examples" by mutating network packets so that the corresponding feature representation (i.e., a NetFlow) was misclassified by the ML-NIDS. Despite the proven effectiveness of such perturbations (which led to correct feature vectors denoting malicious samples that bypassed the targeted ML-NIDS), we argue that *such strategies may not resemble a real attacker's actions.*

Indeed, prior work is limited to applying perturbations by manipulating pre-collected datapoints—such as adding junk bytes to a packet in a PCAP trace. However, from an attacker's viewpoint, *precisely* adding that exact amount of junk bytes to the specific packet (and just that one!) sent by their controlled host may be unfeasible (especially if the payload is encrypted); besides, the packets "seen" by the router (and forwarded to the NIDS) may not correspond to those sent by the attacker-controlled hosts.[4] Put simply, to bypass an ML-NIDS, we argue that a real-world attacker may opt for more straightforward perturbation-related strategies—such as issuing different commands to their controlled hosts. We define such a class of adversarial ML attacks as "host-space perturbations" since they are physically applied to a (attacker-controlled) host.

Research Goal. We seek to: justify that "host-space perturbations" are a realistic (and, so far, overlooked) way to attempt evasion of ML-NIDS; and examine some of their effects.

[1] E.g., changing the values of a "malicious" NetFlow to induce the targeted ML model to classify it as "benign", thereby evading an ML-NIDS [16].

[2] For instance, manipulating the NetFlows may lead to numbers which would represent "impossible-to-generate" network traffic [102].

[3] Most papers on adversarial ML focus on image recognition [14, 111], where "feature-space" perturbations entails manipulating pixels in an image [110].

[4] For instance, consider the attack proposed in [90], described as follows: "a dummy packet is injected into a specified location $k \in [0, n-1]$ among the first n packets of a flow and an [universal adversarial perturbation] is injected into it." An attacker can certainly do so *in theory*—but how, *in practice*?

Focus of the paper. In this work, we consider NIDS and, specifically, those focusing on network traffic analysis. Orthogonal application domains, which are outside our scope, include: IDS for Controller Area Networks, since they use different datatypes (e.g., [33, 99]); or for industrial-control/cyber-physical systems, since they mostly rely on sensor data (e.g., [50, 51, 76]); as well as anti-malware engines [10], or IDS analysing provenance data (e.g. [13, 34, 65]).

3 Host-space Perturbations

We present our primary contribution: the conceptualization of "host-space" adversarial perturbations (HsP). HsP can be seen as an *alternative way* to attack (or assess the robustness of) ML-NIDS. We first introduce HsP with a reflective analysis (§3.1), then define HsP (§3.2) and provide some security considerations on HsP (§3.3).

3.1 Generic Intuition

In security assessments, the researcher should aim at reproducing the workflow that a hypothetical attacker would follow in the real world [14]. In the context of adversarial perturbations, a crucial question that must be answered is: "What is the attacker's problem space?"[5] We answer this question by putting ourselves in the attacker's shoes: through *inductive reasoning* [31], we demonstrate that, in the real world, (most) attackers would (and, likely, can) only operate on the "host space", i.e., on the hosts they control.

Context. Let us assume that an attacker seeks to carry out some malicious operation within a given network. Without loss of generality, such an objective can entail any type of (malicious) network communication: external-to-internal (e.g., a remote DDoS attack); internal-to-external (e.g., data exfiltration); or internal-to-internal (e.g., reconnaissance, lateral movement). The network is protected by some NIDS, which can be deployed anywhere—but, for simplicity, we assume that the NIDS is deployed at the network's border. The NIDS integrates some ML model that analyses network-related data of any type (e.g., packets payload [108], or NetFlows [121]): without loss of generality, we assume that the NIDS analyses NetFlows, since it allows broad coverage (see e.g., [24, 96] or recent surveys [19, 116]). In this context, the attacker performs "actions" on their controlled host (external or internal); such actions lead to the generation of network packets, which are preprocessed into NetFlows, and then fed to an ML model within the NIDS that determines whether the NetFlows are benign or malicious. Hence, in this context, the problem space wherein the attacker operates is defined by the *actions that the attacker can perform on their controlled host*.

Exemplary use-case. Assume the attacker, who has obtained remote access to an internal host, wants to carry out some reconnaissance: the attacker uses the `nmap` command, specifying any parameter (e.g., `-T4 -A`). Such actions generate some packets and NetFlows, whose values depend on the parameters specified by the attacker. This workflow is illustrated in Fig. 2. We observe that the packets are captured by the router/switch (to which the attacker has no access), which forwards them to the NetFlow extractor (to which the attacker has no access), which sends the NetFlows to the ML-NIDS (to which the attacker has no access). Hence, the attacker has some control over the packets sent by their controlled

[5] We provide in Appendix A a discussion on how to tackle this question in adversarial ML domains orthogonal to that of NIDS (e.g., phishing, malware, or computer vision).

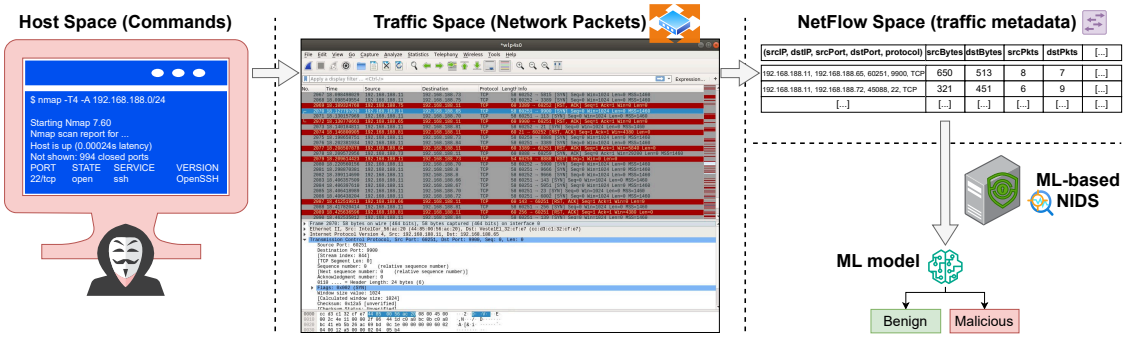


Fig. 2: From attackers' actions to ML inputs. [Left] The attacker launches nmap on their controlled host. [Middle] This leads to the creation of multiple network packets that are captured by some dedicated network appliance (e.g., a router). [Right] Then, NetFlows are extracted from the PCAP trace, which are sent to (and analysed by) the ML-NIDS. A realistic attacker has no access to the "traffic space" (i.e., middle panel) and "NetFlow space" (i.e., right panels): the attacker can only operate at the host level (i.e., left panel). Therefore, directly manipulating the PCAP trace (captured by the router), the NetFlows (extracted from the PCAP trace), or their feature representation (analysed by the ML model) would violate the assumptions of the threat model.

host, but once these packets "leave" this host, our envisioned attacker cannot modify them in any way. Simply put, such an attacker cannot manipulate the data captured by a router/switch, nor the resulting NetFlows, nor the individual feature-vectors analysed by the ML-NIDS—*all of which being ways in which prior work applied adversarial perturbations* (we will justify this assertion in §4). If the attacker suspects that the ML-NIDS is trained to detect reconnaissance attempts issued with nmap -T4 -A, the attacker may take another action, e.g., using -T0 (instead of -T4): doing so would lead to a slower scan, meaning that different packets and, hence, different NetFlows will be generated by their host. Therefore, the perturbations that can be crafted by real-world attackers pertain to the actions (in terms of commands launched) performed on their *controlled hosts*—i.e., the "problem space."

Takeaway. Real attackers perform *actions* leading to changes in the network data generated by their controlled hosts. From an adversarial ML viewpoint, this can be simulated through "host-space perturbations", which require operating at the host level—and not on the data that, despite having been generated by such a host, was captured by an appliance *outside the attacker's control.*

3.2 Definition and High-level Threat Model

We are not aware of any existing definition of "host-space perturbation" (which are conceptually different from the "problem-space transformations" of [87]; we discuss this in §3.3). Hence, we provide our own. Broadly, we define a host-space perturbation as "*any operation that allows an attacker to achieve the same goal by executing commands or actions on the hosts under their control (as defined by the specified threat model)*". Here, a "goal" refers to the successful execution of an attack. A more formal definition is as follows.

Assume an attacker who controls a (set of) hosts $\mathcal{H}$ and aims to achieve a given goal $\mathcal{G}$ through a series of operations $\mathcal{O}$ executed on those hosts. A host-space perturbation is defined as *any alternative set of operations* $\mathcal{O}' \neq \mathcal{O}$ that allows the attacker to achieve the same goal $\mathcal{G}$ by operating on the same set of hosts $\mathcal{H}$. It is implicitly assumed that both $\mathcal{O}$ (and $\mathcal{O}'$) generate network traffic analysed by the ML model that the attacker aims to evade via the host-space perturbation—though evasion is not guaranteed.

We observe that, according to our definition, an HsP does not identify a precise "perturbation".[6] Rather, the HsP is defined as an alternative way to reach the same goal. Note, however, that such a "goal" can be defined either broadly or more narrowly. For instance, a broad goal can be "carrying out a port scan"; a more narrow one is "carrying out a port scan in a low-and-slow manner" [40]: and a very specific one can be "carrying out a port scan in a low-and-slow manner by using nmap". Clearly, a broad goal implies a great number of "alternative ways" in which such a goal can be achieved.

We can hence abstract our generic definition and provide a formal threat model for HsP. To align such a threat model with established works, we follow the guidelines of [14], who recommend defining a threat model by adopting a system-wide view.

Threat Model. Given an attacker having a goal $\mathcal{G}$, which they pursue by using their knowledge $\mathcal{K}$ and capabilities $\mathcal{C}$ on the targeted system to devise a certain strategy $\mathcal{S}$. To realise a valid HsP, the following must be met:

- **Goal.** The overarching goal is the same as in the original threat model (i.e., $\mathcal{G}$), but it must also include evasion of an (ML-based) NIDS.[7] (By "evasion", we intend a misclassification of malicious samples at test/inference-time [14].)
- **Knowledge.** The knowledge is identical to that of the original threat model (i.e., $\mathcal{K}$), but it must be assumed that the attacker expects that the ML-NIDS would detect the strategy $\mathcal{S}$ the attacker would follow if they did not apply any HsP.[8]
- **Capabilities.** The capabilities must not change w.r.t. $\mathcal{C}$: the attacker cannot be assumed to, e.g., control more hosts, or have more privileges on their already controlled hosts.
- **Strategy.** The strategy must change w.r.t. $\mathcal{S}$, and is still dictated by the assumed $\mathcal{K}$ and $\mathcal{C}$, and must only include operations that involve the attacker's controlled hosts.[9]

3.3 Security Considerations

Let us analyse our proposed HsP from a security standpoint (we provide additional "critical" remarks in §7.3).

[6] HsP are *complementary* to traditional adversarial paradigms in the ML-NIDS context.
[7] Otherwise, the attacker would not even attempt any evasion.
[8] Otherwise, why would the attacker even attempt using an HsP?
[9] E.g., an attacker cannot "bribe" an employee to obtain a password; or physically go to a host they wish to portscan in order to check which ports are open locally.

First, our "requirements" on the *goal* and *knowledge* are also implicit in any threat model entailing attacks implemented via adversarial perturbations. For instance, no attacker would, e.g., attempt to introduce a gradient-based perturbation if they did not expect the existence of a detector that would flag their "non-adversarial-examples." Similarly, it is implicit that, when choosing a "new" strategy, attackers would favor simple ones (e.g., guessing-based strategies [14]), which not necessarily are the most optimal ones.

Second, "host-space perturbations" satisfy the required properties of "problem-space adversarial ML attacks" provided by Pierazzi et al. [87]: this is because the perturbation is, by definition, applied in the attacker's problem space. However, and crucially, HsP are not "problem-space *transformations*". Indeed, realising an HsP implies *physically* executing a different set of operations ($\mathcal{O}' \neq \mathcal{O}$). There is no "transformation" in an HsP: the principle is not to "create an adversarial example by applying problem-space transformations to a given (non-adversarial) example"; rather, the point is "creating new datapoints entirely *in the problem space*". And indeed, this is why we emphasized that "evasion is not guaranteed". Put differently, an HsP should be seen as a complementary way through which researchers should investigate the security/robustness of (ML-driven) NIDS *by directly operating in the attacker's problem space* (i.e., on the hosts controlled according to $\mathcal{C}$).

Finally, we further illustrate some intriguing properties of HsP with an example. Let us consider the simple use case (from §3.1) of an attacker whose goal ($\mathcal{G}$) is to "port-scan" some device. The attacker does so by issuing `nmap -T0` (i.e., $\mathcal{O}$) on their controlled host ($\mathcal{H}$). The attacker can also achieve the same goal ($\mathcal{G}$) from the same host ($\mathcal{H}$) by issuing `nmap -T4` ($\mathcal{O}'$). We use this example to make four important considerations on our host-space perturbations.

- *Certain host-space perturbations are easier-to-apply than others.* E.g., changing a parameter from `T0` to `T4` is a simple and valid host-space perturbation. The attacker could also manually probe each port using `netcat`; despite being more labour-intensive, it is a valid host-space perturbation (provided, of course, that `netcat` is available on the attacker's controlled hosts).
- Some host-space perturbations may cause substantial changes in the "feature space", potentially leading to *out-of-distribution* (OOD) samples that are misclassified. However, this is not a concern for the real attacker [14], whose objective is evading the ML-NIDS while achieving their overarching goal (e.g., a port scan in the previous example). HsP should not be seen as ways to induce "minimal" feature-space changes (the focus of most adversarial-ML papers, such as [87]).
- An *invalid host-space perturbation* involves an activity that is not admitted in the threat model—for instance, executing a command that is not assumed to be available on the attacker-controlled host (e.g., running `sudo` without having root privileges), since this violates the envisioned capabilities (i.e., $\mathcal{C}$).
- From a researcher's viewpoint, *it is unwise to simulate the effects of host-space perturbations via manipulations of pre-collected data.* For instance, simulating the change from `-T4` to `-T0` by manipulating the NetFlows (i.e., a "feature-space" perturbation, as done in [16, 94]) can lead to inconsistent adversarial examples [101] (we are not aware of a one-way function that goes from "command" to "NetFlow"); whereas manipulating the PCAP trace (as done, e.g., by [59]) is also unreliable because networks are "unpredictable" ecosystems [19]: for instance, going from `-T4` to `-T0` may lead to some packets being lost or retransmitted.

In our assessment (in §5) we show "easy-to-apply" HsP, as well as of (potentially) "invalid" HsP, and we also show what happens when attempting to approximate HsP (in §6.3).

4 Systematic Literature Review (RQ1)

To scrutinize the novelty of HsP in the adversarial ML context applied to the NIDS domain, we ask ourselves our first research question (RQ1): "Has prior work on the robustness of ML-NIDS considered perturbations involving launching different commands on the attacker-controlled host (i.e., HsP)?" We answer RQ1 with a systematic literature review (SLR).

4.1 Research Method

To answer RQ1, we must fulfill two objectives: *(i)* identify papers investigating the "robustness of ML-NIDS," and *(ii)* inspect their content to determine if the assessment entails HsP.

To tackle the first objective, we adopt an approach similar to recent works [19, 21], which focus on identifying papers from a select subset that is likely to include literature relevant for RQ1. We consider all the works that are "linked" to six pivotal papers: [14, 15, 19, 53, 59, 120]. These works all deal with the subject of ML-NIDS and either *(a)* have been published in top-tier venues [14, 19, 53, 120]; or *(b)* cover a large number of previously-published papers on, among others, "robustness of ML-NIDS" [15, 19, 53]; or *(c)* carry out their evaluations by using perturbation techniques that are methodologically close to HsP [58, 120]. Hence, these works are a valid starting point to select the literature used to answer RQ1.

To tackle the second objective, we adopt a mix of automated keyword-based filtering, and manual qualitative analysis. The latter is done by employing the *dual-reviewer system* (as also done in [14, 21, 95]): two researchers (having [6–9] years of experience in ML-NIDS) inspected the papers and, in case of doubt, discussed their findings to reach a consensus. Altogether, these operations have been done twice: once in August 2024, and a second time in September 2025 (to update our findings with more recent works).

4.2 Systematic Workflow Description

We describe our SLR by following established PRISMA guidelines [85], providing the results after each major step.

Paper collection. First, we consider: the 38 works analysed by [53]; the 46 papers reviewed in [19]; and the 88 papers in [14]. We do so because all these works scrutinised literature related to ML-NIDS (for [19, 53]) or ML security (for [14]) published in top-tier venues within 2017–2023. Then, we consider [15, 59, 120], and apply the snowball method [123], collecting all papers that are cited by (according to Google Scholar) either of these works [15, 59, 120] and which were published until the end of 2024. We do so because these three works [15, 59, 120] are well-known in the ML-NIDS domain and consider "problem space" adversarial ML attacks that resemble real-world scenarios. After removing duplicates, we obtain a set of 316 papers (note: we consider all works returned by Google Scholar, including gray literature, as also done in [95]).

Automated Filtering. We review these 316 papers and determine whether there is an evaluation of "adversarial ML attacks"

against ML-NIDS. We do so by means of a keyword search with the terms "adversarial perturbation / attack / example". We filter the papers mentioning any of the keywords for at least three times in their full text (many papers simply mention similar terms for future work, e.g. [25]), increasing the likelihood that the paper is indeed about adversarial ML; this gives us 201 papers.

Manual Analysis. We analyse the remaining 201 papers that carry out an "adversarial" assessment of ML-NIDS (excluding, e.g., literature reviews [15]). We scrutinise "where" the perturbation is applied: this is done with a qualitative analysis focused on inspecting the experimental methodology adopted in the paper, gauging whether the perturbations stem from attackers issuing different commands on their controlled hosts (i.e., host-space perturbations); we may also check the source code (if provided).[10]

4.3 Findings and Analysis

Across the 201 papers we manually analysed, the perturbations are predominantly crafted by directly modifying the data—potentially before preprocessing (e.g., [7]), or via feature-space manipulations (some of which preserving domain constraints [97]). We mention two noteworthy approaches:

- Catillo et al. [32], explore "problem-space perturbations" by introducing delays through modifications to the attacker's host network configuration using Linux's Traffic Control utility [69]. However, this approach requires the attacker to have admin power on the host, and has the drawback of affecting *all* traffic between this specific host and the target—including the "benign" communications. Under the assumption that the attacker has such a power, this could qualify as a valid HsP. Yet, we could not find any "threat model" or detailed description of the capabilities of the envisioned attacker in [32].
- The adversarial perturbations crafted in [18] are applied to the PCAP trace by using the Python library scapy [29] to add junk bytes to certain network packets. Such an approach is, by definition, *not* an HsP. Although the perturbations are not in the "feature space" (since the targeted ML model analyses NetFlows, whereas the perturbations are applied to network packets) it is unclear whether the corresponding "manipulated" network packets would be received by the involved host exactly in the same order as that reported in the PCAP trace: indeed, appending junk bytes means that the packets are "larger", which leads to more bandwidth being used, and which translates to potential (and ultimately unpredictable) differences in the way the traffic is distributed across the whole network.[11] However, we stress that this approach does preserve domain constraints.

Even papers published in 2025 (outside our SLR) apply direct data manipulations, such as [30] which acknowledge "we cannot guarantee that all the adversarial examples can be produced in a real world setting." Hence, altogether, these findings echo a statement made in a recent work: "Problem-space Evasion Adversarial Attacks are Already Extremely Hard for an Attacker" [47].

Nonetheless, our analysis elucidates a blind spot in ML-NIDS research. To the best of our knowledge, it is unknown how resilient current ML-NIDS are against HsP. We hypothesize that *this gap stems from the practical challenges of studying HsP in a research setting*: it requires access to an operational networked host, executing "adversarial commands," capturing the corresponding traffic, generating the feature representation, and studying the effects on the ML-NIDS. These steps are non-trivial, especially since most prior work relies on public datasets [53] collected in networks (and, hence, hosts) not accessible by downstream researchers.

Answer to RQ1: We could not find any paper that allows to answer positively to RQ1. Hence, according to our SLR (and to our knowledge), no prior work has attempted to "adversarially evade" an ML-NIDS by operating (e.g., launching different commands) on the attacker-controlled host—i.e., our proposed HsP.

5 Demonstration of Host-space Perturbations

Given the negative answer to RQ1, we wonder (RQ2): "what are some effects that HsP can have on existing ML-NIDS?" To tackle RQ2, we carry out a proof-of-concept evaluation of our proposed HsP.[12] Such a demonstration has a twofold goal: *(i)* show how HsP can be practically leveraged (by real-world attackers) and realised (by researchers), and *(ii)* examine their effects on well-known ML-based NIDS (i.e., the crux of RQ2).

5.1 Experiments on Benchmark Datasets

We begin our demonstration by focusing on ML-NIDS trained and tested via the well-known CICIDS17 and CICIDS18 datasets [100]. These datasets, which count over 3k citations on Google Scholar, are widely used to benchmark ML-based NIDS focused on NetFlows. Therefore, they are ideal for our assessment. Importantly: we use the *fixed version* of these datasets (provided in [48] and [72]).

Threat Model. We envision an ML-NIDS that is trained on the data contained in either the CICIDS17 or CICIDS18 datasets. These datasets present similarities: the latter extends the former by including more "benign" data, as well as "malicious" data pertaining to a wider range of attacks. Both datasets have malicious datapoints related to the patator attack, which is a well-known ssh-bruteforcing tool [5]. Specifically, these datasets include network-traffic data captured by launching patator --persistent=1. For an exemplary demonstration, we assume an attacker who, after having compromised a host within the network monitored by the ML-NIDS, wants to ssh-bruteforce a given ssh server while bypassing such an ML-NIDS (i.e., $\mathcal{G}$). In terms of capabilities (i.e., $\mathcal{C}$), the attacker can launch arbitrary commands on their compromised host, but does not have root or physical access to it, and has no control on any other host in the network. Given that the attacker expects the ML-NIDS to be trained to detect patator --persistent=1 (i.e., $\mathcal{K}$), the attacker launches patator --persistent=0 (i.e., the new strategy). Such a strategy entails a valid HsP, which only requires changing a single character of

[10] We also scrutinised if the experiments entailed an original real-world network-data collection step (without, e.g., exclusively relying on benchmark datasets). We found only 12 works (out of 201, i.e., 5.9%) doing so: [7, 8, 25, 32, 63, 81–83, 91, 92, 104, 125]

[11] We reached out to the authors of [18], who confirmed this fact.

[12] Across all our experiments, we do not "data snoop' [21] to inflate the performance of our ML models—we have no incentive to do so. We maintain a clear separation between training and test data, never evaluating on samples seen during training. Moreover, since our data is collected over a short timeframe, a temporal split is unnecessary. Prior empirical studies on similar testbeds have shown that it yields no significant difference/benefit [19]. Additional data-collection and pre-processing details are provided in the Appendix B, whereas complete details of our experiments (e.g., hyperparameters), including the underlying source code, are provided in our repository [6].

Table 1: Experiments on benchmark data. Results (fpr/tpr on benign/malicious NetFlows) of our models for each "train set" and "test set", for both the CICIDS17 and CICIDS18. Boldface denotes NetFlows generated artificially via [117]. We report the std in our repository [6]. We use "P" to denote the option "--persistent" of patator.

Dataset	CICIDS17						CICIDS18		
Train Set	Benign + P=1			Benign + all malicious			Benign + P=1		
Test Set	Benign	P=1	**P=0**	Benign	(all)	**P=0**	Benign	P=1	**P=0**
DT	<0.001	0.997	0.224	<0.001	>0.999	0.214	0.000	1.000	0.000
RF	0.000	0.998	0.490	<0.001	>0.999	0.073	0.000	1.000	0.000
XGB	<0.001	0.998	0.986	<0.001	>0.999	<0.001	0.000	1.000	0.000
SVM	<0.001	0.993	0.000	<0.001	0.997	<0.001	0.000	1.000	0.000
DNN	0.000	0.998	0.000	<0.001	>0.999	<0.001	0.000	1.000	0.000

the command and does not require any additional privilege on the attacker-controlled host (meaning that C does not change).

Baseline Setup. We carry out three experiments—all conforming to our underlying threat model. First, we train five well-known and ML-based classifiers – namely: decision tree (DT), random forest (RF), histogram-gradient boosting (XGB), support vector machine (SVM), and a deep neural network (DNN) – on a training set containing 80% of the benign data in CICIDS17, and on 80% of the data pertaining to patator --persistent=1 in CICIDS17; then, we test such models on the remaining 20% (benign, and of patator --persistent=1) in CICIDS17. Next, we do the same, but for CICIDS18 (thereby obtaining five different models). Next, we train another set of five models, this time using 80% of all malicious data in CICIDS17 (not just the datapoints of patator --persistent=1) alongside 80% of its benign data, and we test these models on the remaining 20% datapoints (benign and malicious). We always repeat this process five times, averaging the results. Altogether, these tests reveal that our models exhibit high tpr (above 0.999) and low fpr (below 0.001), i.e., they align with state-of-the-art performance reported in [48, 72].

Attack implementation and results. To test the HsP, we need to create new data from scratch, conforming to patator --persistent=0. This is because there is no datapoint in either CICIDS17 or CICIDS18 that captures such an attack. To this end, we use the open-source network-simulator tool proposed in [117], which allows users to configure an ad-hoc network environment (which we configured so as to resemble to network environment in CICIDS17 and CICIDS18) and automatically generate labeled data pertaining to attacks launched by a given "attacker" host against a given "target" host (we configured the "attacker" and "target" host so that they are identical to the machines used in CICIDS17/18 to perform the patator --persistent=1 attack, but of course we specified the command patator --persistent=0). This way, we obtained a set of labeled NetFlows that exactly reproduces the behavior of our envisioned attacker. We then submit these NetFlows to the respective ML models, and test their performance. The results of all these experiments are reported in Table 1. We can see that the tpr of our models against the malicious NetFlows of patator --persistent=0 is substantially lower than that achieved against patator --persistent=1. The only exception is the XGB trained only on the malicious NetFlows of patator --persistent=1 on CICIDS17, which obtains tpr=0.986 but which drops to <0.001 when trained on all malicious data of CICIDS17 (this is a clear case of when generalizability across a wide range of malicious attacks reduces the performance against specific attacks). On CICIDS17, the DT and RF are the only models whose tpr is not <0.001, but the resulting values indicate that these models would be bypassed by such an HsP. On CICIDS18, all models are defeated.

5.2 Real-World Network Experiment

The previous experiments were done on "benchmarks", and we had to resort on a dedicated tool to create datapoints that would enable testing an HsP. Here, we assess HsP *on a real-world network*.

Threat Model. We envision a similar threat model as in the previous experiment. However, here, the targeted network is a smart-home network comprising 35+ active hosts, including laptops, desktops, gaming consoles, and various IoT appliances (e.g., light bulbs, smart switches). In this network (which is much larger and recent than that in CICIDS17), two devices are set up so as to enable SSH connections between them. The adversary is assumed to control one of these devices, and seeks to bruteforce his/her way onto the other device by means of the patator tool. The network is monitored by an ML-NIDS that is specifically trained to recognize either patator --persistent=0 (in which case, the attacker would use patator --persistent=1) or patator --persistent=1 (in which case, the attacker would use patator --persistent=0). Note that all such assumptions align with our overarching threat model (in §3.2) while also enabling alignment with the previous experiment (in §5.1).

Baseline Setup. We follow a similar protocol as in the previous experiment. First, we capture a large PCAP trace (of ≈5GB, representing 12h of network activities) from the monitored network: we checked this trace and confirmed that no malicious activities were present. Then, we used the device assumed to be controlled by the attacker (i.e., a laptop running Ubuntu 20.04) to launch patator --persistent=0 against the "targeted" ssh server; we repeated this process also for patator --persistent=1. We captured the traffic, and generated the corresponding NetFlows. We manually labeled such NetFlows, ensuring that only those pertaining to patator were labeled as malicious. Then, we trained our five baseline models (DT, RF, XGB, SVM, DNN): first, we took 80% of the benign data and 80% of the data of patator --persistent=1; then, we trained another set of five baseline models, this time on 80% of the benign data and 80% of the data of patator --persistent=0. We then tested them on the remaining 20%: the performance (both tpr and fpr) was always near-perfect. (As before, we repeated this procedure five times, averaging the results). To further confirm the effectiveness of our models, we even tested them on a larger (benign) PCAP trace of 15GB, confirming that they always yield near-zero fpr, indicating that they are "good" because they do not raise false positives.

Attack results. To gauge the effectiveness of the HsP, we tested the models trained on patator --persistent=0 against the NetFlows produced by patator --persistent=1, and vice-versa. The results of this entire experiment are reported in Table 2. We see that our HsP are very effective: out of the models trained on patator --persistent=1, only one does not have tpr=0: the SVM has tpr=0.49, which is still a substantial drop from the perfect tpr achieved on patator --persistent=1. For the models trained on patator --persistent=0, two (the SVM and the DNN) are not affected, but the remaining three are completely defeated. Note that these results validate those we obtained on the experiments on benchmark datasets: in all cases, changing a single character is enough to bypass most models.

5.3 Additional Variants of HsP

We study other types of HsP. To align these tests with the previous ones, we consider: different ssh-bruteforcing tools (§5.3.1), network-level manipulations (§5.3.2), and system-level changes (§5.3.3). Across

Table 2: Experiments on real-world data. Results (fpr on benign and tpr on attack NetFlows) of our models for each "train set" and "test set" (averaged over 5 trials; we provide the std.dev. in our repository [6]).

Train Set	Benign + P=1			Benign + P=0		
Test Set	Benign	P=1	P=0	Benign	P=0	P=1
DT	<0.001	1.000	0.000	<0.001	1.000	0.000
RF	0.000	1.000	0.000	0.000	1.000	0.000
XGB	0.000	>0.999	0.000	<0.001	1.000	0.000
SVM	0.000	1.000	0.490	0.000	1.000	1.000
DNN	<0.001	1.000	0.000	<0.001	1.000	1.000

these scenarios, we progressively relax the capabilities of the attacker. (The results are in Table 3 in the Appendix B)

5.3.1 ***Different tools.*** There are many ways to carry out ssh-bruteforcing attacks. Beyond patator, one can use well-known tools such as Medusa [3] or Hydra [4]. So, to reproduce a valid HsP, we consider the baseline models (trained on CICIDS17 and CICIDS18 on patator --persistent=1) and test them against NetFlows generated by *Medusa* and *Hydra* (we used, once again, the network-traffic simulator in [117] to do so). Note that these are all valid HsP: these tools do not require additional privileges, and they entail operations that, if successful, enable the attacker to achieve the same objective—violating an SSH server. The results indicate that, for *Hydra*, two models (SVM and DNN) seem to be able to withstand such HsP on both datasets (tpr>0.890), but the remaining three models are always defeated (tpr<0.022); for *Medusa*, the DNN is robust on CICIDS18 (tpr=0.879) and the RF is mediocre on CICIDS17 (tpr=0.658). Still, in all the remaining eight cases, the results indicate that the models are bypassed (the tpr is always below 0.334).

5.3.2 ***Changing the bandwidth.*** We test an HsP similar to the adversarial strategy proposed in [32]. We envision an attacker with high privileges who attempts to bypass the ML-NIDS by inducing network-level changes through tampering with the Traffic Control utility. Such operations can be reproduced via the network simulator in [117], so we used it to generate NetFlows conforming to patator --persistent=1 (i.e., those included in CICIDS17 and CICIDS18) but by varying network-level parameters such as: *latency*=(10ms, 25ms, 100ms), *loss*=(0%, 5%), *corrupt*=(0%, 5%), *duplicate*=(0%, 5%). We assess all 24 combinations of these parameters, each time by launching patator --persistent=1, capturing the corresponding traffic and labeling the NetFlows, and then testing them against the baseline models of CICIDS17 and CICIDS18. We find that only one model is defeated: the RF trained on CICIDS18 (tpr=0). In all other cases, the tpr ranges between 0.729 and 0.993. Hence, we conclude that such HsP have some impact (since the baseline tpr was always >0.999) but do not lead to a complete bypass—despite the fact that manipulating the network channel via Traffic Control requires root privileges (not needed for §5.3.1).

5.3.3 ***Different OS.*** We test an "almost-invalid" HsP. We assume an attacker who, for some reason, can change the entire OS of the host they can control, switching from Ubuntu 16 (the one used in CICIDS17 and CICIDS18) to Ubuntu 18. This is an extreme HsP: the only way to do so in the real world, is if the attacker has physical access to the machine and replaces its OS. While we believe that real-world attackers would opt for other strategies before resorting to such a tactic, we still nonetheless tested the effects of such an HsP (which, we reiterate, assumes a "stronger" attacker than that in §5.1). So, we used the network simulator in [117], thereby creating a machine with Ubuntu 18, and captured its traffic by launching patator --persistent=1. Again, we found that these HsP are effective only against the RF trained on CICIDS18: in all other cases, the tpr remains above 0.75, indicating that these HsP are not very practical to bypass the ML-NIDS (which is only mildly impacted).

Answer to RQ2: Some HsP can drop the tpr of an ML-NIDS from 1 to 0 by changing one character in the command launched by the attacker's controlled host. Even HsP leveraging a different tool but for the same malicious goal can lead to complete bypass. HsP requiring higher privileges are not necessarily more evasive. The effectiveness of HsP against specific ML models varies.

In the Appendix C, we expand our assessment by considering other attack types, models, and datasets.

6 Case Study: Examination of SSH Patator

Our previous results raise a question: "Why is it that, for patator, changing a single character can lead to a complete bypass of the ML-NIDS?" We investigate this phenomenon by examining the data generated by the patator ssh-bruteforcing tool. The goal is twofold: *(i)* provide more insights on how HsP "mutate" the resulting network traffic, producing different input samples that an ML-NIDS may struggle to classify—all with minimal effort from the attacker; and *(ii)* explain why HsP are tough to reproduce by directly manipulating pre-collected datapoints—especially in the feature space.

6.1 Technical Analysis: what does patator do?

We explain how patator carries out its ssh-bruteforcing attempts.

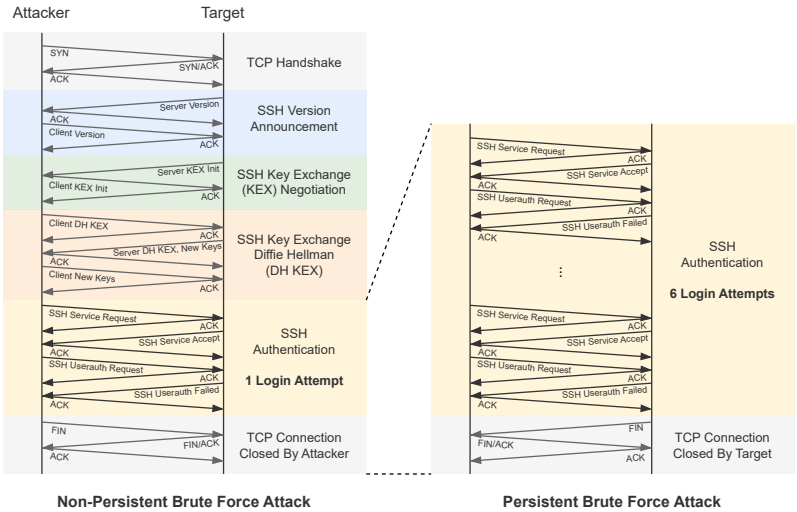
At a high level, patator performs repeated SSH login attempts by trying combinations of usernames and passwords. It can be configured using several options, including the "persistent" flag. When enabled (--persistent=1), patator continues attempting logins over a single TCP connection until the SSH server terminates it (by default, OpenSSH closes the connection after six failed attempts). In contrast, when disabled (--persistent=0), patator initiates a new TCP connection for each login attempt. Fig. 3a shows the operations in both modes, showing the different packet-level interactions between attacker and target with a default OpenSSH setup. While some communication patterns remain unchanged (e.g., the TCP handshake), setting --persistent=1 results in more interactions.

At first glance, one might assume that patator --persistent=1 simply leads to "≈6x more packets" w.r.t. patator --persistent=0. Yet, as we will show, the differences between enabling and disabling the "persistent" flag are more pronounced and less predictable.

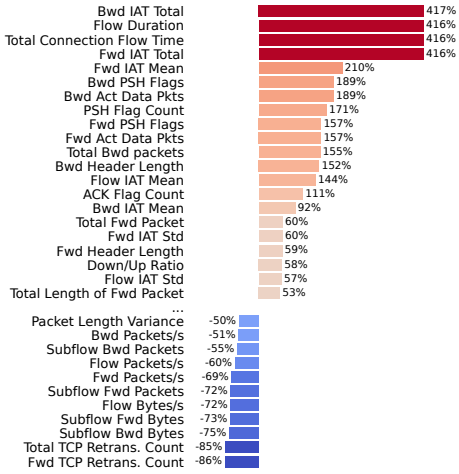
6.2 Real-world Quantitative Analysis

To quantitatively compare patator --persistent=0 and patator --persistent=1, we examine the NetFlows generated by each command. Doing so enables understanding why the ML models trained on one variant may not be able to classify the other "HsP-driven" variant.

We hence take the NetFlows generated in the real-world network (in §5.2), and we compare the various features (computed by CICFlowMeter, for which we used the fixed version in [48]) across the two considered variants of patator. We report in Fig. 3b the ratio



(a) Packet-level. By launching patator -P=0 (left), only one login attempt is made. However, launching patator -P=1 (right) leads to 6x more attempts. This difference is the "packet-level perturbation" induced by an HsP that changes -P=0 to patator -P=1 (i.e., a one-character change in the command launched by the attacker).



(b) NetFlow-level. In a the NetFlow-based feature space, an HsP switching -P=0 to -P=1 leads to substantial differences (numbers in the plot are obtained by computing -P=0 / -P=1)

Fig. 3: Low-level effects of a host-space perturbation. We show what happens when switching from patator --persistent=0 (-P=0) to patator --persistent=1 (-P=1). The former attempts a single login and closes the connection; the latter continues issuing login attempts until the SSH server closes the connection.

of the numerical features between patator --persistent=0 and patator --persistent=1. Specifically, for each NetFlow feature, we aggregate all NetFlows of each variant of patator and then calculate the ratio.

These results align with our explanation (§6.1). For instance, the *FlowDuration* of patator --persistent=0 is much higher (over 4x) than that of patator --persistent=1, because the latter issues multiple attempts (i.e., up to six) for each connection, whereas the former only issues one before closing the connection and initiating a new one (with a new attempt). At the same time, the longer duration also leads to a 75% smaller *Flow Bytes/s*. Put simply, these quantitative results show that these two variants of patator are substantially different at the feature level, explaining why an ML model trained over one variant may not be able to recognize the other variant as being of the same (malicious) class (i.e., an OOD-induced misclassification)). Still, depending on the decision boundaries learned by an ML model during its (unpredictable) training phase, the ML model may still be able to classify a variant of patator as malicious (e.g., the SVM trained only on patator --persistent=0 from the real-world setup was not fooled by patator --persistent=1, see Table 2).

To get a better understanding of the "OOD" potential of an HsP, we report in Fig. 4 (in Appendix B) the t-SNE plot (inspired by [53]) showing the distribution of the NetFlows produced by our two considered variants of patator, as well as that of CICIDS17 and benign NetFlows. We see that samples of patator –persistent=1 (both ours and those in CICIDS17) form a single cluster (validating our implementation); but patator –persistent=0 is a clearly different cluster: therefore, these two variants belong to "different distributions", justifying why ML models may struggle identifying them.

Nonetheless, we make another observation: the differences at the NetFlow level "exceed" our expectations. For instance, patator --persistent=1 also has substantially higher values for the features related to the inter-arrival time (IAT) w.r.t. patator --persistent=0; yet, there does not seem to be much correlation between these higher values (e.g., *Fwd IAT Total* is 416% higher, *Bwd IAT Mean* is 92% higher, *Flow IAT Std* is 57% higher). Moreover, the number of *Ack Flag Count* of patator --persistent=1 is 111% higher than that of patator --persistent=0. Simply put, while we were aware that transitioning from one variant of patator to the other would have led to changes in multiple NetFlow features, we could not anticipate which features and to what extent they would be impacted. Therefore, *we posit that replicating such differences via feature-space perturbations is (almost) impossible* (as we stated in §3.3).

6.3 Pitfall: Approximating HsP

We elucidate what may happen when attempting to approximate HsP by directly manipulating pre-collected datapoints.

6.3.1 ***Manipulating the PCAP.*** Replicating the effects of our HsP by changing the packets within a PCAP trace is, we believe, not humanly feasible. Indeed, there are two cases here:

- *Going from patator --persistent=0 to patator --persistent=1:* this requires adding all the packets of the additional login attempts; but also deleting all the packets of the "new connections".
- *Going from patator --persistent=1 to patator --persistent=0:* this requires to identify all packets of the "persistent" connections, delete them, and create new packets related to new connections.

Practically realising both of the above requires applying precise changes to specific packets, including simulating new (or erasing existing) TCP handshakes and ACKs. All such changes must also account for the overarching network channel (e.g., there may be retransmissions). Hence, we do not see a humanly-feasible way to accurately reproduce a similar HsP by operating on a PCAP trace.

6.3.2 ***Manipulating the NetFlow features.*** Foreseeing the effects of an HsP on the NetFlow features is also challenging; evidence of this are the remarkably different values of the NetFlow features shown in Fig. 3b. However, to provide evidence of "what can go wrong" if one attempts to (incorrectly) reproduce an HsP, we performed a simple experiment. According to our high-level analysis

(§6.1), launching patator --persistent=1 leads to ≈6x more packets (w.r.t. patator --persistent=0) being exchanged. We "naively" simulated this by perturbing the feature space: we took the NetFlows generated by patator --persistent=0 (and patator --persistent=1) in the real-world network, and multiply (divide) the number of packets by 6; we also did so for the number of bytes and updated all other dependent features, ensuring a correct feature vector that preserves domain constraints [101]. Then, we submit these NetFlows, allegedly representing patator --persistent=1 (and patator --persistent=0) to the ML models trained on the "real" variant of the corresponding patator command. Our rationale is that, if these "fictitious" NetFlows resemble that of the "real" patator, we should obtain the same results achieved in §5.2: a perfect *tpr* (see Table 2). However, while this is true for two models (the DNN and the SVM), for three models (DT, HGB, RF) out of five, the *tpr* is *always* 0. This indicates that using such NetFlows for security assessments of ML-NIDS in adversarial environments would be misleading, as they do not represent the attack that would occur in the real world.[13]

Takeaway: Changing a single option (or parameter value) of a malicious command can lead to substantially different network activities—which achieve the same goal, but can cause misclassifications due to OOD (exploitable via HsP). Attempts to reproduce, or approximate, such changes by manipulating pre-collected datapoints is challenging and may result in misleading evaluations.

7 Discussion and Clarifications

We discuss our research, pointing out limitations (§7.1), making ethical remarks (§7.2), and clarifying intrinsic aspects of HsP (§7.3).

7.1 Limitations and Justifications

Our goal (§2.3) is to substantiate the fact that prior work on "adversarial ML attacks against ML-NIDS" focused on perturbation techniques that are hard to realize by real-world attackers—who would rather opt for strategies captured by our notion of HsP.

To reach our goal, we have carried out an SLR (in §4). While we have manually analysed ≈300 papers on this subject, and concluded that no prior work (aside from perhaps [32]) assessed the adversarial robustness of ML-NIDS to HsP, we acknowledge that our paper-selection methodology (based on reputable and recent works) may have led us to overlook some works that did employ techniques which fall in our definition of HsP. Still, even if this is true, it would not undermine our contribution: factually, most prior work on this subject applied perturbations in spaces that are not "reachable" by real-world attackers. We thus elucidate such a complementary "adversarial-perturbation" method to the ML-NIDS community.

Then, to provide evidence of the effects of HsP, we have carried out original experiments (in §5). While such a demonstration entailed developing a total of 125 different models[14] using both data from "synthetic" benchmarks and real-world network environments, we acknowledge that our primary evaluation (in §5 and §6) mostly cover one specific class of HsP, namely, ssh-bruteforcing attacks. Even though we expanded our assessment (in the Appendix) with additional use cases (e.g., port scanning), covering all possible constillations of HsP (in terms of targeted ML-NIDS, considered network environment, and threat models is unfeasible and beyond the scope of a single paper. This, however, does not undermine our conclusions: HsP can impact ML-NIDS in ways that are not captured by perturbations applied to pre-collected datapoints.

Finally, and related to the point above, our evaluation encompasses only one category of ML-NIDS, i.e., those analysing NetFlow features. However, this also does not impact our conclusions: HsP do lead to network-level changes (at both *(i)* the packet level and *(ii)* the NetFlow level). Therefore, HsP are bound to affect any NIDS that falls into our definition (see §2). We do not claim that HsP are "the only", nor a "guaranteed", way to bypass ML-NIDS: therefore, our conclusions are not impacted in the slightest by the scope of our evaluation. Investigating the effects of HsP on all conceivable ML-NIDS would, therefore, just lead to a waste of compute.

7.2 Ethical Considerations and Disclaimers

We do not seek to invalidate or diminish prior research.

It is true that our findings/results reveal that the perturbations considered in prior work may not align with the real world. However, this does not undermine the contributions of prior work. First, because it is still possible to see the corresponding attacks as realistically plausible: this requires assuming a "stronger" attacker—who can, in some way, precisely control/manipulate the data acquired by devices such as routers or the NIDS itself. Second, because (some) prior works did warn that the considered perturbations may not be physically realizable (e.g., [30]). Third, because **it is thanks to such prior work that we were able to elaborate the notion of HsP**. Therefore, prior work was fundamental for our contributions.

For the reason above, we do not seek to examine the experimental approaches adopted in prior work under a critical lens (e.g., to question the alignment between the experimental evaluation and the envisioned threat model). This would not be constructive: prior work (e.g., [120]) relied on methods that, to our knowledge, were the most appropriate ones to support their claimed contributions.

We obtained permission from the owners of the smart-home network to use their data for this research, and also to infect some hosts within their network with malicious tools and launch the corresponding attacks. However, for privacy, we cannot release the PCAP traces—but the NetFlows are included in our repository [6].

7.3 Critical Remarks on HsP (and our response)

In the spirit of a healthy scientific discourse, we report below, in a critique-and-response format, our stance on some legitimate critiques that a reader may have w.r.t. the general idea of HsP.

One can easily manipulate a PCAP trace (e.g., via scapy) and send 'perturbed' packets via tcpreplay. Therefore HsP are not the only way that attackers can use to evade ML-NIDS This is true (note: we never said manipulation of PCAP traces is not possible, nor that HsP are *the only way* to evade ML-NIDS, see §7.1). Attackers can manipulate packet structures directly, but with the risk of breaking

[13] **We do not cheat:** our intention is to use our prior knowledge of patator to derive such an estimate, which is based on what is discussed in §6.1. Clearly, by having knowledge of the quantitative differences between the two considered variants of patator (discussed in §6.2), such a "naive" approach would not make sense. Yet, using such knowledge to craft our feature-space perturbations would be cheating, but also pointless: such knowledge was obtained by generating *real* network traffic, so it is pointless to reproduce the effects by changing the feature space, since it would be sufficient to use the appropriate PCAP trace and generate the corresponding NetFlows.
[14] Given by: 5 classifiers, trained over 5 different training sets (2 from the real world and 3 from benchmarks), repeated for 5 times for statistical robustness.

protocols or leaving obvious artifacts (e.g., unnatural packet headers/payloads). Besides, even if the PCAP manipulation is functionally correct, the PCAP must still be "replayed" (by the host) and the resulting packets captured (by the router—i.e., the device which will forward the data to the ML-NIDS). Nonetheless, instructing a host to reproduce (via tcpreplay) an adversarially-manipulated PCAP trace is an operation that falls into our notion of HsP, as long as it is a permissible operation according to the given threat model (e.g., using tcpreplay requires root privileges, which the attacker may not have). *Nevertheless, the data used for the assessment should be the one captured by the router—and not the one of the modified PCAP trace.* In other words, solely manipulating the packets included in a PCAP trace (assumed to be collected by the router/switch connected to the ML-NIDS) cannot exactly simulate the network activities, "seen" by the router, of the attacker-controlled host.

HsP are not adversarial perturbations! This is an opinion. It is true that the notion of "adversarial perturbation" typically denotes minimal changes in the feature space (e.g., changing one pixel [110] or feature [12]). However, over time, and especially in the security domain, such a notion has changed to encompass attacks wherein the adversary applies some changes to a given "entity" (e.g., an image, a STOP sign, a piece of malware, a website) which is then analysed by some ML model: such changes can be "small' in the respective problem space, but "large" in the feature space (see, e.g., [17]). In the context of HsP, our notion simply denotes a way through which attackers attempt to reach their goal by performing "different" operations on their controlled hosts. The term "different" is what leads HsP to fall into the category of "perturbations": the attacker *expects* that an ML-NIDS analyses the network communications of their controlled hosts, which is why the attacker attempts to reach their goal by perturbing/manipulating/modifying the commands they execute on their host. This is semantically not different from, e.g., applying perturbations that change malware source code (see [87]) or the HTML of a phishing webpage [17]; there are even works whose "adversarial perturbations" lead to "adversarial examples" that are clearly discernible by humans [64]. Therefore, while we acknowledge that HsP can lead to "large" changes to the feature space (see our OOD assessment in §6.3.2), HsP still qualify as "adversarial perturbations" (n.b.: the HsP used in our demonstration mostly entail changing a single character of a command!).

HsP require strong threat models to be staged! This is not true. HsP do not require more knowledge/capabilities than those already envisioned in any given threat model (see §3.2): for instance, an attacker who wants to bypass an ML-NIDS via "gradient-based" perturbations does so because he/she expects the ML-NIDS to detect his/her malicious activities if he/she does not actively attempt to evade the ML-NIDS. We have also explicitly warned (in §3.3) that one should be cautious when experimenting with HsP, as some commands may require privileges that do not conform to a given threat model (in which case, the HsP would not be valid for a security assessment). Finally, we note that all the HsP envisioned in this paper represent threat models wherein the targeted ML system is "invisible" to the attacker (according to [14]).

HsP only apply to ssh bruteforcing! We primarily focused on ssh bruteforcing to provide a comprehensive demonstration (including both ML evaluation and subsequent data-level investigation) to facilitate understanding of HsP, and why they may work. However, HsP can be conceived for any type of malicious network activity. For instance, an attacker can carry out a port scan in a different way (as we described in §3.2), which would be an HsP (which we test in the Appendix). An attacker can also instruct a botnet-infected host to perform its beaconing at different intervals, which would also be a valid HsP. An attacker can also use HsP to achieve the "optimal" feature-space perturbation that evades a given ML model.

8 Recommendations and Outlook

We derive lessons learned (§8.1) and implications to the broader ML-NIDS domain (§8.2), and suggest avenues for future work (§8.3).

8.1 Lessons Learned

We distill three lessons learned from our technical experiments.

Countermeasures. If an ML-NIDS is bypassed via some HsP, such a vulnerability can be addressed via *adversarial training* [98]. Indeed, we have tested this experimentally: for all the ML models we developed in our evaluation (in §5), we re-trained them by augmenting the training set with 80% of the malicious NetFlows generated by any given HsP, and then testing the resulting model on the remaining 20%: the models always obtained $tpr > 0.999$ (and we did not observe any decrease in the baseline performance).

Achieving robustness requires a system-wide mindset. Our experiments show that: *(i)* minor changes to the command used to launch a network-related attack—such as going from `patator --persistent=0` to `patator --persistent=1`; and *(ii)* different implementations of the same network-related attack—such as using *Medusa* or *Hydra* instead of *Patator*; lead to substantially lower detection rates by ML-NIDS trained on a single "variant" of the attack. Essentially, such small changes lead to OOD issues which cannot be countered via ML-centered solutions alone. Indeed, a single application of adversarial training cannot protect against all possible variants of HsP. This highlights that, to comprehensively assess whether a given ML-NIDS can withstand attacks of a specific class (e.g., ssh-bruteforcing attempts) it is necessary to test the ML-NIDS against multiple variants. Such variants must, in theory, encompass all the HsP that the envisioned attacker is allowed to launch—which requires system-wide considerations. Note, however, that some HsP can have no effect whatsoever on the network traffic (and, hence, on the ML-NIDS).[15] We recommend to: provide more specific threat models (e.g., potentially stating, and justifying, what commands are, or not, allowed by the envisioned attacker); but also study the various options that can be used to launch any given malicious command to avoid carrying out potentially redundant experiments. Finally, to better optimize resource allocation, we recommend to examine the mapping between HsP and feature-level changes: this way, one can train the ML-NIDS on a single command to provide robustness against multiple HsP variants of that command.

Reproducing HsP. From the viewpoint of a researcher, practically implementing HsP is challenging. This is because, to ensure correct implementation, it is necessary to operate directly on the host, and capture the corresponding network activities (otherwise, one may create inconsistencies, as shown in §6.3). In our experiments, we adopted two approaches, depending on the use case.

[15] **(Negative result)** E.g., changing the *threads* in patator (e.g., issuing `patator -T=1` or `patator -T=5`) does not lead to any network-level change if the attack targets a single ssh server. We tested this: the tpr never changed, and the NetFlows were not different.

- *Complete "real-world" simulation.* If the researcher has complete control over the network environment from which the network-traffic data is generated, then exploring different HsP is straightforward. In our case (§5.2), we simply launched a different command on the hosts, captured the traffic, labeled it appropriately, and used these datapoints in our ML-focused experiments. However, obtaining access to such a setup requires some effort. A viable alternative is using virtual machines to create a virtual network (some existing datasets have been created in this way [53]); or use specific network-simulation tools (e.g., [61, 117]).
- *Reliance on "benchmark" datasets.* If one wants to assess the robustness of ML-NIDS developed by using data included in a benchmark dataset (e.g., CICIDS17), then it is necessary to adopt a different approach. The data in a benchmark dataset is captured in a specific network; hence, when creating the HsP, the host on which the commands are run must resemble the host used to create the data included in the benchmark. In our case (§5.1), we studied the documentation of CICIDS17/18 and thoroughly examined the data contained in these datasets to derive an appropriate "configuration" that we could use (alongside the network-simulation tool in [117]) to create realistic HsP.

Put differently, implementing HsP requires *expertise in networking*.

8.2 Implications of our Research

We identify three major implications of our findings.

First, our SLR revealed that prior works examining "adversarial ML attacks" against ML-based NIDS carried out their evaluations by considering perturbations that are – while not necessarily unrealistic – hard to practically realise. From the viewpoint of an attacker, the main obstacle to overcome in order to apply such "traditional" perturbations is *finding a way to manipulate datapoints that are collected on dedicated, and highly-secure, appliances* (e.g., the router, or the NIDS itself). Note: we are not claiming that practically evading ML-NIDS cannot be done via, e.g., gradient-based strategies whose results are applied in the feature space; rather, we argue that doing so requires assuming that the attacker has also compromised hosts beyond those that they can remotely control. Hence, future work seeking to study such forms of adversarial ML attacks should justify how the envisioned attacker can perturb pre-collected datapoints.

Second, our experimental evaluation indicated that minimal changes to the malicious commands launched by the attacker can induce substantial alterations in the features analysed by an ML-NIDS, which can lead to evasion. Given the lack of work examining these classes of "adversarial ML strategies", we hence endorse future work to also explore this dimension when assessing the robustness of ML-NIDS. At the same time, we endorse future work to be explicit in providing the commands/options/source-code used to launch any attack included in an experimental evaluation.

Third, the concept of HsP opens the debate of "when do adversarial perturbations stop being an ML-security problem and become a system-security problem?" Indeed, as we discussed (in §7.3), the concept of adversarial perturbations has evolved over the years. Given that ML models are now increasingly integrated into operational systems (including NIDS [19]), we believe that these two domains (ML and system security) should not be disjoint. We posit that studying the security of ML-NIDS should start from a fundamental question: "in what ways can the attacker, within their problem space, *feasibly* influence the data analysed by the ML model?" Indeed, strengthening the considered ML model only against perturbations applied to pre-collected datapoints may lead to a false sense of security if the attacker can achieve evasion by, e.g., merely change a character of a malicious command.

8.3 Future Work

There are several aspects that can be addressed in future work.

First, finding ways to map packet-level (or feature-level) changes to HsP (which is orthogonal to our recommendation in §8.1). For instance, how can an attacker, via HsP, realise the gradient-based attack that would induce a misclassification despite a minimal change in the feature representation of a given sample? This would lead to identifying HsP that are guaranteed to evade the ML-NIDS.

Second, future research can investigate how HsP affect different classes of ML-NIDS, such as those analysing only PCAP traces [120] (which may not be feasible if the traffic is encrypted, and NetFlows would be more informative [119]), or provenance logs [34], or even network graphs [109, 115]. Certain HsP may have no impact whatsoever on the datatype analyzed by some ML-NIDS. For instance, we can reasonably infer that graph-based NIDS may not be impacted by the HsP focused on patator, because the hosts involved in the exchange would still be the same, meaning that the network graph should not be altered. Yet, if the attacker expects the NIDS to rely on graph-based algorithms, the attacker may rely on HsP that alter the network graph (e.g., by triggering connections with other hosts in the network). We believe that our notion of HsP should inspire future work to investigate which types of ML-NIDS can be used to cover potential "blind spots" of certain classes of detectors.

Third, HsP are a way to achieve evasion at test/inference time. Our follow-up adversarial training experiments (§8.1) showed that, by retraining an ML model on the data generated via HsP, it is possible to make such an ML model robust against the specific HsP. However, exhausting all possible combinations of HsP may lead to ML models that, while being effective against the broad class of attacks related to the HsP, may not be robust against other types of attacks. In a sense, this could be a form of "self poisoning", wherein one trades specificity for generalizability. Future work can therefore explore whether there are any tradeoffs: for instance, only a subset of all possible HsP is sufficient to cover the entire spectrum of HsPs of a given attack, and generating new training data would lead to detrimental effects from a generalizability standpoint.

9 Conclusions

We elucidated a blind spot in prior literature on "adversarial ML attacks" against ML-based NIDS.

The potential impact of our proposed host-space adversarial perturbations (HsP) is a call to action. Real attackers favor cheap and simple strategies. Future research should shed more light on this, yet another, security risk of ML-NIDS.

Acknowledgments

We thank the anonymous reviewers for their feedback. This research was partially supported by Hilti, and by the BOSA-AIDE project funded by the Belgian SPF BOSA with reference number 06.40.32.33.00.10. This work originated from a research stay at the University of Liechtenstein and was supported by the Research Foundation – Flanders (FWO) under grant number V450224N.

References

[1] 2014. Wfuzz. https://github.com/xmendez/wfuzz.
[2] 2015. DDoS Slowloris. https://github.com/gkbrk/slowloris.
[3] 2015. Medusa. https://github.com/jmk-foofus/medusa.
[4] 2016. Hydra. https://github.com/vanhauser-thc/thc-hydra.
[5] 2016. patator. https://github.com/lanjelot/patator.
[6] 2026. Our Repo. https://github.com/idlab-discover/HsP.
[7] Maged Abdelaty, Sandra Scott-Hayward, Roberto Doriguzzi-Corin, and Domenico Siracusa. 2021. Gadot: Gan-based adversarial training for robust ddos attack detection. In *IEEE CNS*.
[8] James Aiken and Sandra Scott-Hayward. 2019. Investigating adversarial attacks against network intrusion detection systems in sdns. In *IEEE NFV-SDN*.
[9] Sharmin Aktar and Abdullah Yasin Nur. 2023. Towards DDoS attack detection using deep learning approach. *Computers & Security* 129 (2023), 103251.
[10] Hisham Alasmary, Aminollah Khormali, Afsah Anwar, Jeman Park, Jinchun Choi, Ahmed Abusnaina, Amro Awad, Daehun Nyang, and Aziz Mohaisen. 2019. Analyzing and detecting emerging Internet of Things malware: A graph-based approach. *IEEE Internet of Things Journal* 6, 5 (2019), 8977–8988.
[11] AM Aleesa, BB Zaidan, AA Zaidan, and Nan M Sahar. 2020. Review of intrusion detection systems based on deep learning techniques: coherent taxonomy, challenges, motivations, recommendations, substantial analysis and future directions. *Neural Computing and Applications* 32, 14 (2020), 9827–9858.
[12] Ahod Alghuried, Ali Alkinoon, Abdulaziz Alghamdi, Soohyeon Choi, Manar Mohaisen, and David Mohaisen. 2025. Evaluating the Vulnerability of ML-Based Ethereum Phishing Detectors to Single-Feature Adversarial Perturbations. *ACM Distributed Ledger Technologies: Research and Practice* (2025).
[13] Abdulellah Alsaheel, Yuhong Nan, Shiqing Ma, Le Yu, Gregory Walkup, Z Berkay Celik, Xiangyu Zhang, and Dongyan Xu. 2021. ATLAS A sequence-based learning approach for attack investigation. In *USENIX SEC*.
[14] Giovanni Apruzzese, Hyrum S 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 *SaTML*.
[15] Giovanni Apruzzese, Mauro Andreolini, Luca Ferretti, Mirco Marchetti, and Michele Colajanni. 2021. Modeling realistic adversarial attacks against network intrusion detection systems. *ACM Digital Threats: Research and Practice* (2021).
[16] Giovanni Apruzzese and Michele Colajanni. 2018. Evading botnet detectors based on flows and random forest with adversarial samples. In *IEEE NCA*.
[17] Giovanni Apruzzese, Mauro Conti, and Ying Yuan. 2022. SpacePhish: The Evasion-Space of Adversarial Attacks against Phishing Website Detectors Using Machine Learning. In *Proc. ACSAC*.
[18] 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*. 149–160.
[19] Giovanni Apruzzese, Pavel Laskov, and Johannes Schneider. 2023. SoK: Pragmatic assessment of machine learning for network intrusion detection. In *IEEE European Symposium on Security and Privacy (EuroS&P)*.
[20] Giovanni Apruzzese, Pavel Laskov, and Aliya Tastemirova. 2022. SoK: The impact of unlabelled data in cyberthreat detection. In *EuroS&P*.
[21] 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*.
[22] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket.. In *NDSS*.
[23] Tim Bai, Haibo Bian, Abbas Abou Daya, Mohammad A Salahuddin, Noura Limam, and Raouf Boutaba. 2019. A machine learning approach for RDP-based lateral movement detection. In *IEEE LCN*.
[24] Osama Bajaber, Bo Ji, and Peng Gao. 2024. P4control: Line-rate cross-host attack prevention via in-network information flow control enabled by programmable switches and ebpf. In *IEEE S&P*.
[25] Diogo Barradas, Nuno Santos, Luís Rodrigues, Salvatore Signorello, Fernando MV Ramos, and André Madeira. 2021. FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications.. In *NDSS*.
[26] Dipkamal Bhusal, Md Tanvirul Alam, Monish K Veerabhadran, Michael Clifford, Sara Rampazzi, and Nidhi Rastogi. 2024. PASA: Attack Agnostic Unsupervised Adversarial Detection using Prediction & Attribution Sensitivity Analysis. In *IEEE European Symposium on Security and Privacy*.
[27] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *ECML PKDD*.
[28] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition* (2018).
[29] Philippe Biondi. 2025. Scapy. https://scapy.readthedocs.io/en/latest/.
[30] Dominik Brockmann, Eric Lanfer, Nikolas Wintering, and Nils Aschenbruck. 2025. Now You See Me/Now You Don't: Constrained Adversarial Attacks in Network Intrusion Detection Across Datasets and Machine Learning Models. In *2025 IEEE 50th Conference on Local Computer Networks (LCN)*. IEEE, 1–9.
[31] Thomas E Carroll, David Manz, Thomas Edgar, and Frank L Greitzer. 2012. Realizing scientific methods for cyber security. In *LASER Workshop*.
[32] Marta Catillo, Antonio Pecchia, Antonio Repola, and Umberto Villano. 2024. Towards realistic problem-space adversarial attacks against machine learning in network intrusion detection. In *ARES*.
[33] Paolo Cerracchio, Stefano Longari, Michele Carminati, Stefano Zanero, et al. 2024. Investigating the impact of evasion attacks against automotive intrusion detection systems. In *Symposium on Vehicles Security and Privacy (VehicleSec)*.
[34] Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. 2024. Kairos:: Practical Intrusion Detection and Investigation using Whole-system Provenance. In *IEEE S&P*.
[35] Antonio Emanuele Cinà, Kathrin Grosse, Ambra Demontis, Battista Biggio, Fabio Roli, and Marcello Pelillo. 2024. Machine learning security against data poisoning: Are we there yet? *Computer* (2024).
[36] CloudFlare. 2020. One more (Zero Trust) thing: Cloudflare Intrusion Detection System. https://blog.cloudflare.com/one-more-zero-trust-thing-cloudflare-intrusion-detection/.
[37] CloudFlare. 2024. *DDoS threat report for 2023 Q4*. Technical Report. CloudFlare. https://blog.cloudflare.com/ddos-threat-report-2023-q4/
[38] Christopher Crowley. 2025. *SANS 2025 SOC Survey*. Technical Report. SANS Research Program. https://www.sans.org/white-papers/sans-2025-soc-survey
[39] Cybersecurity Insiders. 2025. Pulse of the AI SOC Report 2025. https://www.cybersecurity-insiders.com/pulse-of-the-ai-soc-report-2025-from-alert-fatigue-to-actionable-intelligence-how-ai-is-reshaping-detection-response-and-analyst-confidence/.
[40] Darktrace. 2018. How Darktrace Finds 'Low and Slow' Cyber Threats. https://www.darktrace.com/blog/flying-under-the-radar-how-darktrace-detects-low-and-slow-cyber-attacks.
[41] Hervé Debar, Marc Dacier, and Andreas Wespi. 1999. Towards a taxonomy of intrusion-detection systems. *Computer networks* 31, 8 (1999), 805–822.
[42] Hervé Debar, Marc Dacier, and Andreas Wespi. 2000. A revised taxonomy for intrusion-detection systems. In *Annales des Télécommunications*.
[43] Dorothy E Denning. 1987. An intrusion-detection model. *IEEE TSE* (1987).
[44] Alec F Diallo and Paul Patras. 2024. Sabre: Cutting through Adversarial Noise with Adaptive Spectral Filtering and Input Reconstruction. In *IEEE S&P*.
[45] Rohan Doshi, Noah Apthorpe, and Nick Feamster. 2018. Machine learning DDoS detection for consumer internet of things devices. In *IEEE S&PW*.
[46] Laurens D'hooge, Miel Verkerken, Bruno Volckaert, Tim Wauters, and Filip De Turck. 2022. Establishing the contaminating effect of metadata feature inclusion in machine-learned network intrusion detection models. In *DIMVA*.
[47] Mohamed ElShehaby and Ashraf Matrawy. 2026. A novel perturb-ability score to mitigate evasion adversarial attacks on flow-based ML-NIDS. *JISA* (2026).
[48] Gints Engelen, Vera Rimmer, and Wouter Joosen. 2021. Troubleshooting an intrusion detection dataset: the CICIDS2017 case study. In *IEEE SPW*.
[49] ENISA. 2023. *Enisa Threat Landscape*. Technical Report. ENISA. https://www.enisa.europa.eu/topics/cyber-threats/threats-and-trends
[50] Alessandro Erba, Andres F Murillo, Riccardo Taormina, Stefano Galelli, and Nils Ole Tippenhauer. 2023. On Practical Realization of Evasion Attacks for Industrial Control Systems. In *RICSS Workshop*.
[51] Alessandro Erba, Riccardo Taormina, Stefano Galelli, Marcello Pogliani, Michele Carminati, Stefano Zanero, and Nils Ole Tippenhauer. 2020. Constrained concealment attacks against reconstruction-based anomaly detectors in industrial control systems. In *Annual Computer Security Applications Conference (ACSAC)*.
[52] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Robust physical-world attacks on deep learning visual classification. In *IEEE CVPR*.
[53] Robert Flood, Gints Engelen, David Aspinall, and Lieven Desmet. 2024. Bad Design Smells in Benchmark NIDS Datasets. In *IEEE EuroS&P*.
[54] Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. 2021. Realtime robust malicious traffic detection via frequency domain analysis. In *ACM CCS*.
[55] Francesco Fusco, Xenofontas Dimitropoulos, Michail Vlachos, and Luca Deri. 2012. pcapIndex: an index for network packet traces with legacy compatibility. *ACM SIGCOMM Computer Communication Review* 42, 1 (2012), 47–53.
[56] Sebastián García, Alejandro Zunino, and Marcelo Campo. 2014. Survey on network-based botnet detection methods. *Secur. and Commun. Netw.* (2014).
[57] Mengmeng Ge, Xiping Fu, Naeem Syed, Zubair Baig, Gideon Teo, and Antonio Robles-Kelly. 2019. Deep learning-based intrusion detection for IoT networks. In *IEEE PRDC*.
[58] Dongqi Han, Zhiliang Wang, Wenqi Chen, Ying Zhong, Su Wang, Han Zhang, Jiahai Yang, Xingang Shi, and Xia Yin. 2021. Deepaid: Interpreting and improving deep learning-based anomaly detection in security applications. In *ACM CCS*.
[59] 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. *JSAC* (2021).
[60] Qingying Hao, Nirav Diwan, Ying Yuan, Giovanni Apruzzese, Mauro Conti, and Gang Wang. 2024. It Doesn't Look Like Anything to Me: Using Diffusion Model to Subvert Visual Phishing Detectors. In *USENIX Sec*.

[61] Ahmad Hariri, Murat Yuksel, and David Mohaisen. 2024. RL-Based Speculative Installation of Unseen Flows in SDNs for Low-Latency Applications. In *IEEE ICMLCN*.

[62] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J Doug Tygar. 2011. Adversarial machine learning. In *AISec Workshop*.

[63] Steve TK Jan, Qingying Hao, Tianrui Hu, Jiameng Pu, Sonal Oswal, Gang Wang, and Bimal Viswanath. 2020. Throwing darts in the dark? detecting bots with limited data using neural data augmentation. In *IEEE S&P*.

[64] Fujiao Ji, Kiho Lee, Hyungjoon Koo, Wenhao You, Euijin Choo, Hyoungshick Kim, and Doowon Kim. 2025. Evaluating the effectiveness and robustness of visual similarity-based phishing detection models. In *USENIX SEC*.

[65] Zian Jia, Yun Xiong, Yuhong Nan, Yao Zhang, Jinjing Zhao, and Mi Wen. 2024. MAGIC: Detecting advanced persistent threats via masked graph representation learning. In *USENIX SEC*.

[66] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. 2019. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity* (2019).

[67] Platon Kotzias, Kevin Roundy, Michalis Pachilakis, Iskander Sanchez-Rola, and Leyla Bilge. 2023. Scamdog Millionaire: Detecting E-commerce Scams in the Wild. In *ACSAC*.

[68] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2017. Adversarial machine learning at scale. *ICLR* (2017).

[69] Alexey N. Kuznetsov. 2014. tc(8) — Linux manual page. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html.

[70] Wenke Lee and Salvatore J Stolfo. 2000. A framework for constructing features and models for intrusion detection systems. *ACM TiSSEC* (2000).

[71] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. 2013. Intrusion detection system: A comprehensive review. *JNCA* (2013).

[72] 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 Conference on Communications and Network Security*. IEEE.

[73] Ruofan Liu, Yun Lin, Xianglin Yang, Siang Hwee Ng, Dinil Mon Divakaran, and Jin Song Dong. 2022. Inferring phishing intention via webpage appearance and dynamics: A deep vision based approach. In *USENIX Security Symposium*.

[74] Majed Luay, Siamak Layeghy, Yash Pandey, Gayan Kulatilleke, and Marius Portmann. 2025. Multimodal LLMs for Zero-Shot Intrusion Detection Using NetFlow Visualisations. In *IEEE LCN*.

[75] Keane Lucas, Weiran Lin, Lujo Bauer, Michael K Reiter, and Mahmood Sharif. 2024. Training Robust ML-based Raw-Binary Malware Detectors in Hours, not Months. In *ACM CCS*.

[76] Yuan Luo, Ya Xiao, Long Cheng, Guojun Peng, and Danfeng Yao. 2021. Deep learning-based anomaly detection in cyber-physical systems: Progress and opportunities. *ACM Computing Surveys (CSUR)* 54, 5 (2021), 1–36.

[77] 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*.

[78] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. In *NDSS*.

[79] Emily A Nack, Morgan C McKenzie, and Nathaniel D Bastian. 2024. Aci-iot-2023: A robust dataset for internet of things network security analysis. In *MILCOM*.

[80] Maryam M Najafabadi, Taghi M Khoshgoftaar, Clifford Kemp, Naeem Seliya, and Richard Zuech. 2014. Machine learning for detecting brute force attacks at the network level. In *IEEE BIBE*.

[81] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2021. Defeating {DNN-Based} traffic analysis systems in {Real-Time} with blind adversarial perturbations. In *USENIX Security Symposium*.

[82] Samuel Ndichu, Tao Ban, Takeshi Takahashi, Akira Yamada, Seiichi Ozawa, and Daisuke Inoue. 2024. Adversarial Evaluation of AI-Based Security Alert Screening Systems. In *IEEE CyberSciTech*.

[83] Satoshi Okada, Houda Jmila, Kunio Akashi, Takuho Mitsunaga, Yuji Sekiya, Hideki Takase, Gregory Blanc, and Hiroshi Nakamura. 2024. XAI-driven adversarial attacks on network intrusion detectors. In *EICC*.

[84] Ahmed Fawzi Otoom, Emad E Abdallah, et al. 2023. Deep learning for accurate detection of brute force attacks on IOT Networks. *Proc. Computer Science* (2023).

[85] Matthew J Page, Joanne E McKenzie, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, et al. 2021. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. *BMJ* 372 (2021).

[86] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. 2018. Sok: Security and privacy in machine learning. In *IEEE EuroS&P*.

[87] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. 2020. Intriguing properties of adversarial ml attacks in the problem space. In *IEEE Symposium on security and privacy (SP)*.

[88] Michal Piskozub, Fabio De Gaspari, Freddie Barr-Smith, Luigi Mancini, and Ivan Martinovic. 2021. Malphase: Fine-grained malware detection using network flow data. In *ACM AsiaCCS*.

[89] Mahdi Rabbani, Leila Rashidi, and Ali A Ghorbani. 2024. A graph learning-based approach for lateral movement detection. *IEEE TNSM* (2024).

[90] Amir Mahdi Sadeghzadeh, Saeed Shiravi, and Rasool Jalili. 2021. Adversarial network traffic: Towards evaluating the robustness of deep-learning-based network traffic classification. *IEEE TNSM* (2021).

[91] Ola Salman, Imad H Elhajj, Ayman Kayssi, and Ali Chehab. 2022. Mutated traffic detection and recovery: an adversarial generative deep learning approach. *Annals of Telecommunications* 77, 5 (2022), 395–406.

[92] Pedro Miguel Sánchez Sánchez, Alberto Huertas Celdrán, Gérôme Bovet, and Gregorio Martínez Pérez. 2024. Adversarial attacks and defenses on ML-and hardware-based IoT device fingerprinting and identification. *FGCS* (2024).

[93] Tinshu Sasi, Arash Habibi Lashkari, Rongxing Lu, Pulei Xiong, and Shahrear Iqbal. 2024. An efficient self attention-based 1D-CNN-LSTM network for IoT attack detection and identification using network traffic. *JIT* (2024).

[94] Madeleine Schneider, David Aspinall, and Nathaniel D Bastian. 2021. Evaluating model robustness to adversarial samples in network intrusion detection. In *IEEE International Conference on Big Data (Big Data)*.

[95] Saskia Laura Schröer, Giovanni Apruzzese, Soheil Human, Pavel Laskov, Hyrum S Anderson, Edward WN Bernroider, Aurore Fass, Ben Nassi, Vera Rimmer, Fabio Roli, et al. 2025. SoK: On the offensive potential of AI. In *SaTML*.

[96] Christoph Sendner, Jasper Stang, Alexandra Dmitrienko, Raveen Wijewickrama, and Murtuza Jadliwala. 2024. Mirageflow: a new bandwidth inflation attack on tor. NDSS.

[97] Giorgio Severi, Simona Boboila, Alina Oprea, John Holodnak, Kendra Kratkiewicz, and Jason Matterer. 2023. Poisoning network flow classifiers. In *Annual Computer Security Applications Conference*.

[98] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. 2019. Adversarial training for free! *NeurIPS*.

[99] Md Hasan Shahriar, Wenjing Lou, and Y Thomas Hou. 2023. CANtropy: Time series feature extraction-based intrusion detection systems for controller area networks. In *Proc. of Symp. on Vehicles Security and Privacy (VehicleSec)*. 1–8.

[100] Iman Sharafaldin, Arash Habibi Lashkari, Ali A Ghorbani, et al. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* (2018).

[101] Ryan Sheatsley, Blaine Hoak, Eric Pauley, Yohan Beugin, Michael J Weisman, and Patrick McDaniel. 2021. On the robustness of domain constraints. In *CCS*.

[102] Ryan Sheatsley, Blaine Hoak, Eric Pauley, and Patrick McDaniel. 2023. The space of adversarial strategies. In *USENIX Security Symposium*.

[103] Nathan Shone, Tran Nguyen Ngoc, Vu Dinh Phai, and Qi Shi. 2018. A deep learning approach to network intrusion detection. *IEEE TETCI* (2018).

[104] Renato S Silva and Luís M Felipe de Moraes. 2024. GonoGo-Assessing the Confidence Level of Distribute Intrusion Detection Systems Alarms Based on BGP. *IEEE TNSM* (2024).

[105] Software Analyst Cyber Research. 2024. Revolutionizing Security Operations: The Path Toward AI-Augmented SOCs. https://softwareanalyst.substack.com/p/revolutionizing-secuity-operations.

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

[107] Ruoyu Song, Muslum Ozgur Ozmen, Hyungsub Kim, Raymond Muller, Z Berkay Celik, and Antonio Bianchi. 2023. Discovering adversarial driving maneuvers against autonomous vehicles. In *USENIX SEC*.

[108] Wenguang Song, Mykola Beshley, Krzysztof Przystupa, Halyna Beshley, Orest Kochan, Andrii Pryslupskyi, Daniel Pieniak, and Jun Su. 2020. A software deep packet inspection system for network traffic analysis and anomaly detection. *Sensors* (2020).

[109] Wenjia Song, Hailun Ding, Na Meng, Peng Gao, and Danfeng Yao. 2024. Madeline: Continuous and low-cost monitoring with graph-free representations to combat cyber threats. In *ACSAC*.

[110] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2019. One pixel attack for fooling deep neural networks. *IEEE TEC* (2019).

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

[112] Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin, and Wei-Yang Lin. 2009. Intrusion detection by machine learning: A review. *Exp. Syst. Appl.* (2009).

[113] 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 Symposium on Security and Privacy (SP)*.

[114] Emmanouil Vasilomanolakis, Shankar Karuppayah, Max Mühlhäuser, and Mathias Fischer. 2015. Taxonomy and survey of collaborative intrusion detection. *ACM computing surveys (CSUR)* 47, 4 (2015), 1–33.

[115] Andrea Venturi, Matteo Ferrari, Mirco Marchetti, and Michele Colajanni. 2023. ARGANIDS: a novel network intrusion detection system based on adversarially regularized graph autoencoder. In *ACM SAC*.

[116] Miel Verkerken, Matisse Callewaert, Laurens D'hooge, Tim Wauters, Bruno Volckaert, and Filip De Turck. 2025. RustiFlow: Bridging the Gap Between Security Research and Practice using eBPF-based Network Flow Extraction. In

WTMC (IEEE EuroS&P).
[117] Miel Verkerken, Laurens D'hooge, Bruno Volckaert, Filip De Turck, and Giovanni Apruzzese. 2026. ConCap: Practical Network Traffic Generation for (ML- and) Flow-based Intrusion Detection Systems. In *IEEE SaTML*.
[118] Hung Nguyen Viet, Quan Nguyen Van, Linh Le Thi Trang, and Shone Nathan. 2018. Using deep learning model for network scanning detection. In *ICFET*.
[119] Gernot Vormayr, Joachim Fabini, and Tanja Zseby. 2020. Why are my flows different? a tutorial on flow exporters. *IEEE ComSurvTut* (2020).
[120] 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).
[121] Xian Wang. 2022. Enidrift: A fast and adaptive ensemble system for network intrusion detection under real-world drift. In *ACSAC*.
[122] Feng Wei, Hongda Li, Ziming Zhao, and Hongxin Hu. 2023. {xNIDS}: Explaining deep learning-based network intrusion detection systems for active intrusion responses. In *USENIX Sec.*
[123] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *EASE*.
[124] Fabian Woitschek and Georg Schneider. 2021. Physical adversarial attacks on deep neural networks for traffic sign recognition: A feasibility study. In *IV*.
[125] Haonan Yan, Xiaoguang Li, Wenjing Zhang, Rui Wang, Hui Li, Xingwen Zhao, Fenghua Li, and Xiaodong Lin. 2023. Automatic evasion of machine learning-based network intrusion detection systems. *IEEE TDSC* (2023).
[126] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. 2017. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access* (2017).

Appendix A On the "spaces" of perturbations

To further illustrate the differences between the "spaces" of adversarial perturbations, we describe three exemplary scenarios that, despite involving domains orthogonal to NIDS, enable a better understanding of the crux tackled by our work.

Attacks vs Autonomous Driving Cars. Assume a car that relies on ML to analyse the surrounding environment so that proper driving decisions can be made. An exemplary use case is an ML model that must recognize *traffic signs*. In this case, the "feature space" is represented by the pixels of the images acquired by the sensors of the car (e.g., cameras), which are provided as input to the ML model to discern whether *(i)* the image contains a traffic sign, and *(ii)* if so, which sign it is—so that the car can react accordingly.

To introduce a perturbation in such a feature space (i.e., direct pixel manipulation of the digitally-acquired image), the attacker can **(A)** tamper with the software embedded in the car's system—which is doable, but impractical unless the attacker has compromised the systems of the car's manufacturer, or manipulated the systems of the targeted car [124]. The attacker, however, can also **(B)** manipulate the traffic sign *in the physical world*. To do this, the attacker can, e.g., tamper with a specific traffic sign, so that whenever some sensors capture images of such a traffic sign, the resulting image will (ideally) fool the classification of the ML model [52, 107].[16]

N.b.: **(A)** and **(B)** denote two different threat models where the attacker's goal is the same, but the capabilities intrinsically change due to a semantically different problem space: in **(A)**, the problem space overlaps with the feature space; in **(B)**, there is no overlap.

Attacks vs Malware Detectors. Assume a malware detector that relies on ML to analyse some applications and determine whether they are malicious or not. An exemplary use case is the DREBIN detector for Android malware [22]: as input, DREBIN receives a feature vector of thousands of binary features, each related to a specific functionality of the corresponding Android app (e.g., permissions).

To *reliably* introduce perturbations in such a feature space, the attacker must have access to the preprocessing mechanism that extracts the feature vector (which will be analysed by the ML model, i.e., DREBIN), or otherwise be able to change its output. Doing so means that the attacker has root access of either *(a)* the Android device executing the anti-malware app, or *(b)* of a remote security system. Both of these cases are possible, but assume a very powerful attacker who has already compromised the security system.

Alternatively, the attacker can *manipulate the source-code* of the (malicious) Android app, so that the resulting feature vector will have values that bypass the ML model [75, 87]. This is easier to do by an attacker, since they have complete control over the source code of the piece of malware that they develop; however, in doing so, the attacker must be careful not to implement changes that would hinder the malicious functionality of the resulting app.

Attacks vs Phishing Website Detectors. To counter phishing websites mimicking benign websites, state-of-the-art solutions rely on ML models for visual recognition. For instance, PhishIntention [73] extracts the logos from the screenshot of a given webpage, and then uses ML to determine if such logos resemble those of a well-known brand (e.g., PayPal); then, if a match is found, the systems checks if the domain of the analysed webpage matches the domain of the well-known brand (ideally, phishing webpages resemble well-known brands, but are hosted under different domains). In this case, the feature space is represented by the image of a logo (extracted from the screenshot of a webpage).

To introduce perturbations in such a feature space, the attacker simply needs to visually alter the logo in the phishing webpage—on which they have, by definition, complete control. For instance, the attacker can take the logo image, change some pixels, and stitch the new "adversarial" logo onto the phishing webpage [60].

This is an exemplary case in which *the attacker may have complete control on the feature space*, because the attacker's problem space "overlaps" with the ML-model's feature space. However, there are other types of phishing website detectors (reliant, e.g., on the analysis of information extracted from the HTML) that do not allow an attacker to tamper with the feature space—unless the attacker has access to the internal workflow of the targeted phishing detection system. We point the interested reader to [17, 67].

Takeaway. Adversarial perturbations can be applied in many ways and in many "spaces". Yet, depending on the specific application domain, introducing some perturbations may be challenging for real-world attackers. Hence, *it is crucial to identify the "problem space" in which the attacker is allowed to operate.* In such a way, it is possible to define what actions an attacker can take to reach their underlying goal—and, hence, simulate plausible adversarial strategies, studying their effects, and devise countermeasures.

Appendix B Data Collection and Preprocessing

We explain the data collection and preprocessing for our tests. We also report some experimental results (Table 3 and Fig. 4).

[16] This example is useful to explain a "host-space" perturbation: instead of manipulating the image, which is acquired and processed by devices that are not under the attacker's control (i.e., a perturbation to a piece of data), the attacker creates the "perturbation" by interacting with the physical world (i.e., which, in our case, is the attacker's host).

Table 3: Additional security assessments. We report the *tpr* achieved by the baseline models trained on benchmarks (CIC17, CIC18) against the NetFlows conforming to different host-space perturbations

Model	Hydra		Medusa		Network change		OS change	
	CIC17	CIC18	CIC17	CIC18	CIC17	CIC18	CIC17	CIC18
DT	0.019	0.000	0.000	0.000	0.743	0.729	0.750	0.750
RF	0.018	0.000	0.658	0.000	0.993	0.000	0.849	0.000
XGB	0.021	0.000	0.005	0.000	0.883	0.729	0.956	0.750
SVM	0.890	0.895	0.214	0.334	0.749	0.733	0.750	0.750
DNN	0.941	0.902	0.221	0.879	0.822	0.749	0.768	0.750

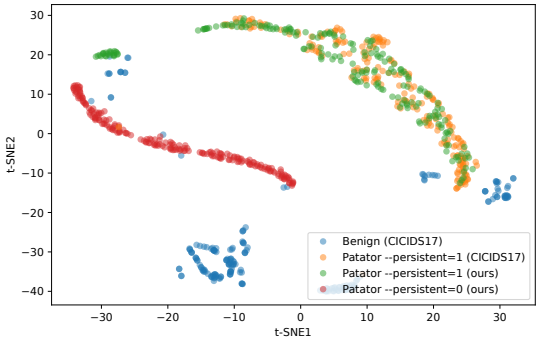


Fig. 4: OOD verification. We use a tSNE plot to visualize the distribution of NetFlows generated by patator (as well as benign ones).

B.1 Real-world Smart-home network capture

The network environment contains 40–50 physical devices. These include smartphones, laptops/desktops, gaming consoles, and various IoT devices (e.g., smart speakers and lightbulbs). All devices connect to a router via a WiFi 5 or 2.4 interface. The router has an Internet connection with 50 Mbps download and 5 Mbps upload. The router is a FritzBox 7590, and a (local) Raspberry Pi4 with PiHole serves as DNS server, forwarding queries to Cloudflare. Devices on the network receive security patches and their owners are security experts, so it is reasonable to treat the traffic as "benign" (and even if some traffic is "malicious", it belongs to a different class than patator, so our conclusions remain unaffected). To implement our attack we used two laptops, both running Ubuntu 18.04: the host from which the attack was launched mounts an Intel i7-7700H with 32 GB of RAM; whereas the host acting as the target mounts an Intel N4100 with 8 GB of RAM.

For our experiments, three PCAP traces were captured: two consisting of benign background traffic, and one related to the patator SSH brute-force attacks. The background traffic was first captured on 6 November 2023 and a second time on 26 August 2024—the same day the malicious attacks were executed and captured. The benign traffic from the 26 August capture totals 6 GB for 8M packets (yielding nearly 50K NetFlows), whereas the attack capture measures 45MB for 241k packets (corresponding to 8k NetFlows). The (benign) trace captured in November 2023 is 17GB for 17M packets, resulting in 30k NetFlows (extracted via CICFlowMeter)

B.2 Handling the CICIDS17/18 Datasets

The CICIDS17 and CICIDS18 [100] benchmark datasets have been known to have various issues for years (see [48, 53, 72]). We are aware of this, which is why we always used the "fixed" version of these datasets, following the methods discussed in [48] (for CICIDS17) and [72] (for CICIDS18) to ensure that our experimental setup was correct and that labelling issues, if present, were minimal. After downloading the source PCAP data for these datasets, we extracted the NetFlows via the (fixed) version of CICIFLowMeter, and applied the labeling logic in [48, 72]. We removed the "attempted" NetFlows, since they are not useful for our purposes.

Then, we cleaned the resulting NetFlows from features that have historically been known to create shortcuts during the training stage of ML models. Specifically, following the recommendations of prior work [20, 21, 46], we remove the following features: "id", "Flow ID", "Src IP", "Dst IP", "Timestamp", "FWD Init Win Bytes", and "Bwd Init Win Bytes". Then, we associate the source and destination ports to the IANA port-type categories (and then encoded as 0, 1, or 2 for respectively *well-known*, *registered*, and *dynamic*). Finally, we sanitized all missing values and removed any duplicate NetFlow.

Appendix C Additional Experiments

We describe additional experiments we have carried out to further expand our assessment and provide a broader understanding of the potential effects of HsP. Particularly, we provide further evidence that HsP can lead to "OOD" samples that cause misclassifications.

Disclaimer. The spectrum of possile attacks and corresponding HsP is virtually infinite. We cannot cover all such possibilities. Here, we merely want to provide additional insight on the potential of HsP-driven attacks to bypass ML-NIDS.

C.1 HsP against DeepLearning-based NIDS

In the main paper, we considered "traditional" detectors created using the scikit-learn toolkit. Here, we expand our assessment by considering additional detectors based on more complex (and recent) deep-learning (DL) techniques and frameworks.

Objective and Method. We want to see if "advanced" DL-based detectors can also be fooled by the same HsP considered in our primary experiment (§5.1). We consider three classes of DL-driven detectors. Two are inspired by [122], which carried out evaluations considering a DL-based *autoencoder* (drawn from [78, 103]), as well as a *recurrent neural network* (drawn from [63, 126]). The third is an application of a *large-language model* (LLM), inspired by [74, 117].

Implementation. For the autoencoder (AE), we use PyTorch to replicate the 6-layer AE in [122] (also used in [78]). For the recurrent neural network (RNN), we tried to replicate long-short-term-memory method in [63], but we ultimately opted for a gated-recurrent unit classifier because it was faster to train. For the LLM, we use ChatGPT 5.4 (the best available as of March 2026). We use the same one-shot prompting technique in [117] where the LLM is told to classify NetFlows in malicious/benign while being provided with "training" data to so that it can determine some boundaries between benign and malicious datapoints; we also tried a full zero-shot prompting strategy, which did not yield good results (the LLM classifies everything as benign). The code for both the RNN and the AE, as well as the prompts for the LLM, are in our repository [6].

Evaluation. To align this experiment with that in our main paper (in §5.1), we consider the (fixed) CICDS17 dataset, and consider its benign samples alongside the malicious ones of patator --persistent=1 to develop our DL-based classifiers; we split this dataset in train:test with an 80:20 split. Then, we submit the HsP-variant of the considered attack (patator --persistent=0) and see how the DL-based classifiers react. We report the available results in Table 4. We can see that these more complex models have a very good baseline

Table 4: Experiments on additional DL-based classifiers.

	tpr (baseline)	tpr (HsP)	fpr
AE [122]	0.9983	0.0000	0.0026
RNN [122]	0.9899	0.0000	<0.0001
LLM [117]	0.9949	0.0000	<0.0001

performance (high tpr with low fpr) but cannot recognize our HsP as malicious. Note that we do not seek to generalize this conclusion: we simply show the impact of HsP on other types of classifiers.

C.2 HsP entailing different Attacks

Our main paper (in §5) mostly considered HsP focusing on ssh-bruteforcing attacks (e.g., Medusa, Hydra, or patator). Here, we expand our assessment by showing other use-cases of HsP.

Attacks. We consider *fuzzying* (via wfuzz [1]), *port-scanning* (via nmap), and *ddossing* (via slowloris [2]). We consider these attacks because they align with the setup of our primary experiments (in §5): the (fixed) CICIDS17 dataset contains port-scanning (via nmap), ddossing (via slowloris), and fuzzying (implemented via custom-made scripts for web attacks—which can be replicated via wfuzz). So, our choice of attacks allows creating valid HsP that can be assessed against the classifiers considered in our main paper (but trained on the "baseline" variant of the attack included in CICDS17). We assume an attacker with the same knowledge and capabilities as that in our primary evaluation (§5.1), but the goal is clearly different (i.e., fuzzying/ddossing/portscanning instead of ssh-bruteforcing).

Implementation. We consider the five classifiers (DT, SVM, RF, DNN, HGB) and train them on 80% of the benign NetFlows of CICIDS17, and 80% of the NetFlows available for the fuzzying, ddos, and portscanning attacks in CICIDS17; we then test these classifiers on the remaining 20% to assess their performance in the absence of HsP. Then, we craft HsP by using the same network-simulator tool used in our main paper [117], configured in the same way used for the experiments on CICIDS17 (§5.1). Specifically, for portscan, we consider several variants of nmap; for fuzzying, we consider five variants of wfuzz; for ddos, we consider six variants of slowloris. We report the specific commands in the Supplementary material.

Evaluation. The results are shown in Table 5. For ddossing, the classifiers are always defeated. For fuzzying, DT, RF, and HGB retain limited recall, while SVM and DNN collapse completely. For portscanning, there are cases in which the HsP have little effect, because they were similar to the variant of nmap used in CICIDS17 (which, e.g., considered the options -sS or -sT [100], and was likely launched with the default -T3). It is hence expected that such HsP do not have a substantial impact on the classifiers. However, some variants still successfully bypass our detectors.

C.3 Assessments on a different Benchmark

In our paper, we tested HsP against ML models trained on data from a real-world network (§5.2) and on two well-known benchmark datasets: CICIDS17 and CICIDS18 (§5.1). Here, we show the applicability (and impact) of HsP on a completely different setup.

Setting and Threat Model. We consider a network resembling a typical Internet-of-Things environment. It encompasses various smart devices, such as smart devices (e.g., smart bulbs or smart switches, or smart doorbells), cameras, but also laptops and servers. The network is protected by an ML-NIDS that receives network-traffic data collected from the router in NetFlow format; the ML-NIDS is trained to detect some DDoS and SSH-bruteforcing attacks. We assume an attacker who has infiltrated in this network by compromising one host, and seeks to launch DDoS attacks from the inside, or attempt SSH bruteforcing attempts (the attacker's knowledge and capabilities hence align with the assumptions in our main paper §5). The attacker seeks to reach their goal by means of HsP, hoping to bypass the detection of the ML-NIDS.

Table 5: Experiments on more attacks. We test HsP related to portscanning (nmap), fuzzying (wfuzz), and DDoS (slowloris). These attacks are included in CICIDS17.

Model	Wfuzz			Nmap			Slowloris		
	tpr (baseline)	tpr (HsP)	fpr	tpr (baseline)	tpr (HsP)	fpr	tpr (baseline)	tpr (HsP)	fpr
DT	0.9524	0.2000	0.0476	0.9985	0.6912	0.0022	0.9993	0.0000	0.0017
RF	1.0000	0.2000	0.0476	0.9996	0.6324	0.0004	1.0000	0.0000	0.0000
HGB	0.9524	0.2000	0.0476	0.9996	0.7059	0.0007	1.0000	0.0000	0.0000
SVM	1.0000	0.0000	0.0000	0.9922	0.5735	0.0101	0.9990	0.0000	0.0003
DNN	1.0000	0.0000	0.0000	0.9970	0.6324	0.0037	0.9997	0.0000	0.0007

Experimental setup. To implement such a use case, we consider the CIC-BCCC-NRC TabularIoTAttack-2024 benchmark dataset [93]; particularly, we focus on the partition discussed in ACI-IoT-2023 [79]. This partition contains labeled NetFlow data (generated via CICFlowMeter 3.0) of an IoT network resembling that of our envisioned threat model. Among the malicious NetFlows, some are generated via hydra (which is an ssh-bruteforcing tool [4]—also used in our main paper §5.3.1) and some are generated by slowloris (which is a tool for launching DDoS attacks [2]). So, after *(i)* studying the documentation in [79] we *(ii)* use the network traffic generator in [117] to replicate a setup aligning with that in [79]—such that the hosts involved in the malicious communications are aligned with that of the authors of [79].[17] Then, we *(iii)* use our setup to launch HsP related to our two considered attacks. Specifically:

- for hydra, we used: hydra .L /$Path/top-usernames-shortlist.txt -P /$Path/2023-200_most_used_passwords.txt ssh://$IP -t 10
- for slowloris, we change the default script (available in [2]) to use HTTPS mode, i.e., by launching python3 ./slowloris.py --https

N.b.: the objective of this proof-of-concept experiment is to investigate the impact of some HsP on a different network environment.[18]

Evaluation. We train the same classifiers used in our main paper on the benign and malicious samples of our considered attacks in [79], reserving 20% of the available NetFlows for testing purposes; then, we test them on our HsP variants of these attacks. The results are in Table 6. Our classifiers attain near-perfect tpr against the "baseline" variant of each of our attacks, while also exhibiting low fpr for the tree-based models and somewhat higher fpr for SVM/DNN. However, against our HsP-variants of hydra, only DT is not affected: all other classifiers recognize these NetFlows as benign. For slowloris, all classifiers are defeated. We can therefore conclude that HsP can be dangerous even in a different setup.

Table 6: Experiments on different benchmark. We test HsP of hydra [4] and slowloris [2] on the [79] section of CIC-BCCC-NRC TabularIoTAttack-2024 [93].

Model	Hydra [4]			Slowloris [2]		
	tpr (baseline)	tpr (HsP)	fpr	tpr (baseline)	tpr (HsP)	fpr
DT	1.0000	1.0000	0.0016	1.0000	0.0000	0.0003
RF	0.9984	0.0000	0.0008	1.0000	0.0000	0.0000
HGB	0.9984	0.0000	0.0016	1.0000	0.0000	0.0003
SVM	0.9828	0.0000	0.0408	0.9946	0.0000	0.0188
DNN	0.9953	0.0000	0.0071	0.9995	0.0000	0.0038

[17] E.g., we set a network channel of 100Mbit/s with no packet loss because this is typical in a relatively-small network environment, especially given that our attacks entail internal-to-internal communications.

[18] Unfortunately, due to lack of documentation, we were unable to identify the exact command used to launch hydra and slowloris in [79]. Nonetheless, we have reason to believe that our "custom" commands are different from those used by the creators of [79]. For instance, by default, hydra sets -t 16 whereas we use -t 10 (see [4])