Classify Me Correctly if You Can: Evaluating Adversarial Machine Learning Threats in NIDS

Neea Rusch[1], Asma Jodeiri Akbarfam[1], Hoda Maleki[1], Gagan Agrawal[2](✉), and Gokila Dorai[1]

[1] Augusta University, Augusta, GA, USA
[2] University of Georgia, Athens, GA, USA
gagrawal@uga.edu

Abstract. Network intrusion detection systems (NIDS) are increasingly developed using machine learning (ML) techniques. However, incorporating ML into NIDS introduces a new vulnerability: the threats and limitations arising from adversarial machine learning (AML) attacks. Specific to this application domain, AML could enable an attacker to disguise incoming malicious packets and fool a NIDS to classify them as benign. Although AML has been researched actively in other domains, assessing its impact in networks remains an outstanding challenge, especially since network protocols pose a *constrained domain* for adversarial packet generation. More specifically, there is a need to experiment with the latest advances in AML attacks – usually developed for an unconstrained domain – on such domains. This paper presents a novel approach to this problem, where a variety of attacks could still be applied and correctly evaluated in a constrained domain. We show an implementation of this approach for NIDS, by developing an adversarial packet validator for different network protocols. By conducting extensive experiments using multiple data sets, ML models, and attacks, we show how our approach can bridge the gap between progress in AML and a constrained domain like NIDS. Evaluation enabled by our approach and its implementation suggests that black-box evasion attacks continue to be a threat to NIDS, despite many constraints offered by this domain.

Keywords: Adversarial Machine Learning · Network Security · Network Intrusion Detection Systems · Evasion Attacks · Network Traffic Analysis

1 Introduction

Network intrusion detection systems (NIDS) is an area where machine learning (ML) has been utilized effectively in the past [1,12]. By learning *signatures*

This research is supported by NSF Grant No. 2131509 and NSA Grant No. H98230-21-1-0163.

H. Duan et al. (Eds.): SecureComm 2023, LNICST 567, pp. 3–22, 2025.
https://doi.org/10.1007/978-3-031-64948-6_1

associated with malicious packets, NIDS provide a critical additional layer of security through ML. A variety of models—particularly ones based on deep learning [1]—have been developed and deployed successfully [16, p. 158].

One of the recent interesting developments in machine learning has been around *adversarial machine learning* (AML) [15]. AML involves many techniques and tools [26], but the most relevant techniques for intrusion detection domain are *evasion attacks*. The objective of evasion is to find inputs to ML models that result in misclassification, allowing attacker to remain undetected by the system's security mechanism [15]. AML research includes developing *attacks*, or search methods, to increasingly effectively find inputs that produce misclassification; and developing *defenses* to enhance model robustness against such attacks. As these techniques have evolved, mature software toolkits are now available that incorporate large number of attacks, defenses, and metrics [25].

AML attack techniques pose significant risks to ML-based NIDS systems [12,15,38]. For example, using an evasion attack, an adversary can send a *malicious* packet that will bypass the NIDS since the model will classify it as *benign*. Regrettably, the intersection of AML and ML-based NIDS remains largely under-explored. Most work in AML has been in the context of image processing and the associated task of object recognition [15]. A domain like object recognition exemplifies an *unconstrained domain*, since all image pixels are candidates for perturbation during adversarial example generation. In comparison, network domain is not only a *constrained domain*, but also adds the following distinct considerations to adversarial example generation.

Constrained Domain Restricts Acceptable Perturbations. In a network domain, adversarial example generation cannot apply perturbations universally [2,11,31]. There are many constraints on fields, corresponding to how only a structurally *valid packet* can travel over a network. This has two separate implications, as many fields are not *mutable*. Those that are, may have complex correlations between other fields that a valid packet must satisfy. These nuanced relationships are ignored by algorithms applying universal perturbations during adversarial example generation. On the other hand, the number of efforts to generate adversarial packets for a constrained domain is limited [2,31].

Traditional Evaluation Metrics are Inapplicable. Advanced AML techniques are primarily designed for image domains, where human perception offers a relevant quality metric. Perturbations that are undetectable to human observer are preferable. For this reason, attack algorithms are designed to minimize the distance between original and adversarial examples, typically using an ℓ_p-norm [15]. However, when data is not intended for observation, distance metrics are not meaningful to assess the quality of adversarial examples [31]; and other considerations, elaborated below, become important.

Model Invocations Must Be Limited. Most adversarial algorithms query the target model interactively, often without restrictions, during adversarial example generation. This cannot be done in network domains. If an attacker sends several malicious packets, the network administrator is expected to block the IP address, especially if they are correctly detected as malicious. Therefore, limiting model queries is necessary when applying adversarial attacks in network domains.

Misclassification is Class Sensitive. Unlike many other domains, there is a significant distinction in the impact of misclassification labels. The security implications differ for misclassifying a benign packet as malicious and misclassifying a malicious packet as benign. The former can degrade system availability and cause overhead, but the latter can compromise system integrity.

Based on the above observations, this paper presents a new approach, and a detailed experimental study, bridging the gap between AML advances and NIDS. Unlike the limited amount of existing work that generates adversarial examples for a constrained domain [2,31], our approach allows for the numerous existing efforts on unconstrained domains to be applied and tested on NIDS. This paper makes the following contributions.

(1) We describe an approach to construct an *evaluation system*, which leverages advanced AML techniques from unconstrained domains, and identify the key metrics to perform risk assessments of realistic evasion attacks on NIDS.
(2) We introduce a rule-based network packet *validator* for several common network protocols. It specializes the proposed evaluation system to *constrained* network environments. The validator is generalizable and extends to additional protocols and constraints.
(3) We provide a concrete *implementation* of the proposed evaluation system design. Using state-of-the-art classifiers, adversarial attacks and defenses, we report the results of an experimental evaluation performed on two network data captures. Our evaluation is open source and available for reuse.
(4) Our evaluation study shows that despite many constraints imposed by the domain, adversarial attacks continue to pose challenges for NIDS systems. Even the state-of-the-art defense mechanisms have at best mixed success.

2 Background

2.1 Efficient Intrusion Detection with Machine Learning

Handling network security operations in a dynamic environment and at scale requires deployment of efficient and intelligent tools. These tools need automatic capabilities to correctly inspect traffic flows and to quickly respond to identified anomalous behaviors. Network intrusion detection systems (NIDS) are deployed with a goal to protect an entire network and its devices from intrusions. They operate by scanning traffic and raising alarms of potential security violations and breaches. ML methods are increasingly used to train NIDS because they have demonstrated efficiency in detecting potential threats [1]. NIDS are trained using network data captures to predict traffic instances as benign (normal) or malicious (attack). Various learning algorithms have been used to train NIDS, with deep neural networks and autoencoders becoming particularly popular in recent years [1,19]. But the inclusion of ML techniques also exposes NIDS to new vulnerabilities in the form of adversarial machine learning [34,35].

2.2 The Capabilities of Adversarial Machine Learning

Adversarial machine learning, or AML, aims to design algorithms capable of resisting security challenges. This involves studying attacker capabilities, understanding consequences of adversarial attacks, and designing methods to defend against attacks [26]. An adversary's objective is to exploit a vulnerability of a model, obtain insight on its construction, or alter its behavior. If a system protects valuable assets—financial services, secure networks, automated machinery, etc.—presence of adversarial vulnerabilities poses real-world security risks [5]. Conversely, adversarial defenses protect systems by reducing adversarial attack surfaces.

Adversarial attacks differ in characteristics, aims, and based on where they occur in the model training and deployment pipeline. *Training phase attacks* acquire or directly influence data during model generation. Prime examples are *poisoning attacks* that contaminate training data with adversary's own examples to teach model specific behaviors [12]. *Testing phase attacks* target existing models and aim to recover data or evade detection [26]. For example, *inference attacks* recover model's training data, and *model extraction attacks* discover the structure or parameters of a model. *Evasion attacks* exploit imperfections of a trained model by altering inputs to avoid correct output classification. Specifically, the goal of evasion is *misclassification*, which occurs when a record's classification label changes due to applied adversarial perturbations [26].

An important aspect of an adversarial attack is its associated *threat model*, which describes attack context and adversary's goals, knowledge, and capabilities [5,12]. There are two primary kinds: a *white-box* threat model that assumes attacker has complete knowledge of the target system, and a *black-box* threat model that assumes internal mechanics of the target system are inaccessible [5]. However, even in a block-box model, an adversary can query the model's input/output behavior, either through an oracle or generated surrogate model [17]. Black-box attacks represent more realistic attack scenarios [12].

Similar to attacks, adversarial defenses can be applied at different phases of model training and deployment. In this context, *robustness* refers to model's ability to withstand adversarial attacks. Possible defense strategies include pre- and post-processing, which smoothen adversarial perturbations or hide meaning of model outputs [37,39]. Training-based defenses purposely include adversarially generated examples in the training data, to teach the model malicious behaviors [33]. Detection-based defenses raise alerts of suspicious inputs in the inference phase during prediction [18]. Adversarial defense mechanisms have been studied extensively for linear models and neural networks, but results are still limited for other models [4].

2.3 Motivations for Analyzing AML Risks in NIDS

AML research has traditionally focused on unconstrained domains, where all input elements can be perturbed. In the context of a constrained setting like NIDS, this is problematic because universal perturbation range overestimates adversary's capabilities, and potentially underestimates capabilities under constrained domains [31]. Assessments of threats in constrained domains are largely unavailable. However, evaluation of adversarial threats and potential mitigations are relevant for all domains where ML models are being deployed.

In the meantime, sophisticated software tools have emerged that have enabled testing modern classifiers, attacks, and defenses. These enable experimenting and prototyping various AML techniques with reduced software development overhead. A few examples include Adversarial Robustness Toolbox [25], AdverTorch [9], Foolbox [28] and CleverHans [7]. Although these solutions vary slightly in scope and available features, broadly their goal is the same: to facilitate running AML attacks on ML models. Consistent with most existing research on AML, these tools also extensively focus on unconstrained domains.

Our goal is to bridge the gap between a large volume of existing and ongoing research, including availability of toolkits as described above, and constrained domains. By leveraging modern software tools – with suitable adjustments – we can study impact of AML on NIDS. This allows to rapidly reuse state-of-the-art techniques and evaluate impact of AML in previously under-investigated cases. We hope to extract new insight of network threats, and that our exploration serves to inspire the development of extended AML techniques and improve security of ML-based NIDS.

3 Our Approach and Its Implementation

The crux of our approach is the following hypothesis:

> Constraints of a domain can be captured through certain rules, which can be applied to filter adversarial records generated by a method for an unconstrained domain, resulting in successful adversarial records for the constrained domain.

If this hypothesis holds true, we will have an approach for launching attacks in a constrained domain while benefiting from the large body of ongoing efforts on unconstrained domains.

To validate the hypothesis, we show that (i) this approach can successfully generate valid adversarial attacks to evade state-of-the-art classifiers, including those incorporating methods to make them robust, (ii) constraints of the domain can be captured through simple rules, and (iii) the process can still be efficient despite the use of unconstrained methods and then having the additional step of filtering away packets that do not meet constraints of the domain.

Our proposed validator uses a set of first-order rules to classify adversarial records as *valid* or *invalid*, i.e., whether they satisfy the constraints of the domain or not. Despite its simplicity, there are several considerations that make the validator interesting and a critical part of the system. First, the validator enables the use of sophisticated attacks—including those that assume full control over feature space—and allows to transfer advancements from other domains to experiments on NIDS. Second, the need to identify and enforce constraints is a significant barrier to applying adversarial techniques in constrained domains. Thus, it is significant that with a validator, integrating constraints into an automated evaluation system is indeed achievable.

3.1 An Example Implementation of the Approach

We describe the implementation of our proposed approach in the context of NIDS. In the considered scenario, a NIDS is trained on a binary classification problem, to label incoming network traffic flows as benign or malicious. The adversary's goal is to cause misclassification in the system by applying AML evasion techniques. Misclassification is *untargeted*: changing the classification label in either direction malicious ⇄ benign counts as success. However, the direction of label change impacts the target system differently. Misclassification from malicious to benign is more harmful, as it implies the attacker has successfully evaded detection by NIDS and may compromise system integrity.

The system has five core components as illustrated in Fig. 1.

Classifier – A NIDS classification model is trained using supervised learning without retraining. The model is trained eagerly to provide fast lookup times, because it must serve high-throughput traffic with low latency.

Input data – Data sets are traffic captures from different kinds of networks. They include various network protocols, modern traffic patterns, and multiple attack classes. Captures are preferably observed in realistic networks and on real hardware devices. We partition the input data randomly into training and test sets during a preprocessing phase.

Threat model – Black-box evasion attacks are within scope. Adversary cannot access model internals but can perform limited queries against it. The query limit is defined as an experiment configuration option. The information returned by a query is determined by an attack algorithm.

Defense strategy – The model can be enhanced with a training-phase defense. The defense can be enabled or disabled by experiment configuration.

Packet validator – A rule-based packet validator works to identify structurally legal perturbations such that a packet can be transmitted over a network. It flags and drops non-permissible instances.

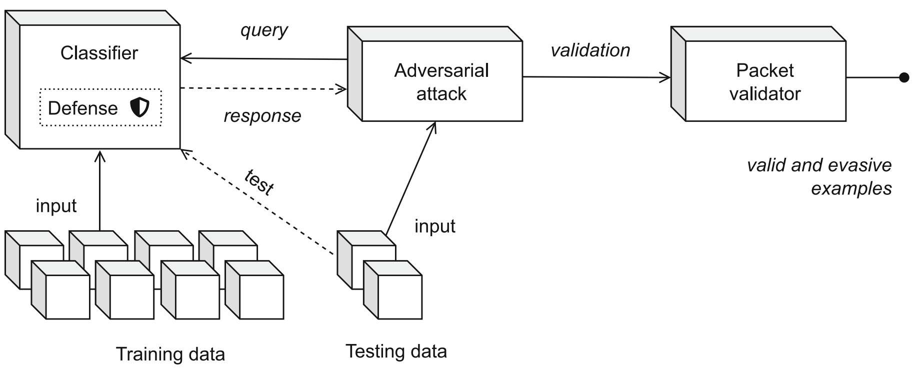

Fig. 1. System components and their interactions.

The system follows factorial experiment design. This allows to compare executions of two system replicas, that vary on one parameter, to measure the impact of that parameter. The parameter to compare can be, e.g., data size, query limit, classifier, attack or defense activation.

We use the system to conduct an adversarial risk assessment as follows. We identify a target model that operates as a network security mechanism inside a NIDS. We train the model, using a choice supervised classification algorithm, on a training data set obtained from partitioning the input data. A defense mechanism can be optionally activated during this training phase. We validate the model against a test set. Next, we assume the role of an adversary, to apply a choice black-box evasion attack. The attack is applied without modification to its algorithm, even if the attack permits control over the entire feature space. The attack input is a copy of the test set. Before the attack, we identify immutable attributes, i.e., those that cannot be modified for a packet to be structurally valid. The immutable attributes are either frozen for the duration of the attack, if attack generation method supports it; or restored to their original values after the attack. The adversary may query the target model as specified by the attack algorithm, but within a configurable query restriction. Finally, we evaluate the adversarially generated examples, using a packet validator, to identify and keep the ones permissible under network domain constraints. Adversarial success rate is measured on examples that are initially classified correctly, produce misclassification following the attack, and pass the validation phase.

3.2 Learning and Applying Domain Constraints

Network data has multiple challenges regarding identifying domain constraints. These include: (i) some attributes are properties of transmission (e.g., packet round-trip time) and thus immutable, (ii) mutability may be directional and allowed to sender or receiver only, (iii) single attribute can be a compound of other fields, making its relational constraints and dependencies difficult to identify, or (iv) attribute's full range may not occur in the input data, and thus not algorithmically learnable.

Considering potential approaches to capture domain constraints, we rejected approximate rule-based learning algorithms. In rule-based learning, a learner operates by evaluating a set of if-then clauses to determine the validity of a hypothesis [20, p. 274]. Although many algorithmic approaches exist for constructing such learners—Learn-One-Rule [20, p. 278], Ripper [32], etc.—we found these to be insufficient. Identifying network constraints requires domain expertise to ensure feature dependencies are strictly maintained. Approximate algorithms do not provide such guarantees.

In our approach, we identify constraints by inspection of input data, and group them by network protocols that occur in the data. It is reasonable for an adversary to deploy a similar approach, even without knowledge of system internals and with limited input examples, because network protocols have established public schemas. We show examples of discovered rules in Table 1. Observe that they are allowed to contain variables, are specialized by protocol, and target numerous features. With this construction, it is possible to craft expressive hypotheses and validate arbitrarily complex feature dependencies.

Table 1. Examples of packet validation rules. They contain variables and vary by data set features and protocol. Violation of any rule makes a record inadmissible.

Protocol	Constraint ⇒ Rule
TCP	Round-trip time is sum of sync and ack ⇒ $\texttt{synack} + \texttt{ackdat} = \texttt{tcprtt}$
TCP	Record start time does not exceed last time ⇒ $\texttt{stime} \geq \texttt{ltime}$
UDP	All TCP fields are zero ⇒ $\texttt{swin} = 0 \wedge \texttt{dwin} = 0 \wedge \texttt{stcpb} = 0 \wedge \ldots$
ICMP	Connection attempt seen but no reply, response packets and bytes are zero ⇒ if $\texttt{conn\_state} = \texttt{S0}$ then $\texttt{resp\_pkts} = 0 \wedge \texttt{resp\_ip\_bytes} = 0$

We define the rule evaluation procedure by the record protocol. For a given record and its respective protocol, every applicable rule is evaluated. A valid record must satisfy the conjunction of all applicable rules. The acceptance policy is permissive: a record is valid if there is no reason to reject it. This is because rejecting a valid record would discard an adversarial example that is interesting for system evaluation. However, accepting an invalid record is not equally harmful: if an adversary attempted to send it, it would simply be dropped by the network. This policy also means the packet validator acceptance rate represents the upper bound of true valid records.

3.3 Metrics for Evaluation

Domains that impose constraints or yield imperceptible data require quality metrics that differ from conventional standards like ℓ_p-norm. In network domain, the need for different metrics arises from properties relating to, e.g., model queries and input size restrictions; and the significance and impact of misclassification, since different inputs produce different responses in NIDS. We identify the following evaluation metrics.

Query efficiency. Networks deploy security mechanisms to quickly block rogue senders who initiate traffic at high frequency or in large quantities. The number of times an adversary can query a model undetected is limited. In our evaluation context, query efficiency refers to the ability to generate adversarial examples with minimal model invocations. Intuitively, an attack that requires fewer queries is preferable, because its execution is more realistic.

Input-size efficiency. Like query efficiency, restricting input size is relevant for an adversary to remain undetected. Input size may also be restricted due to unavailability of examples. The question of interest then is, if only a small initial sample of inputs can be obtained or used, is the adversary successful at executing an attack? A successful attack using a smaller input-size is preferable over another that requires a larger input size.

Class-label distribution. For some adversarial scenarios, deception of any kind is equally preferable. For network data this assumption is false. Classification output labels vary in significance, because they induce different behavior in NIDS. Particularly, the ability to convert malicious packets to benign is preferable over the opposite, because its impact is more damaging. Therefore, class label distribution rate gives a comparable metric for system evaluation.

Success rate. We evaluate the number of times an adversary is successful at misclassification while satisfying domain constraints. Specifically, we report the number of examples that are misclassified and valid, out of the total number of examples. This gives a comparable metric as a fraction of adversarially successful examples.

4 An Illustrative Evaluation Setup

We concretely implemented the evaluation system, to experimentally measure performance of state-of-the-art adversarial techniques in NIDS using the identified metrics. The implementation consists of:

- 2 data sets: traditional and IoT traffic and modern attack flows.
- 2 classifiers: tree-based ensemble learner and deep neural network.
- 2 defenses: classifier-specific and applied at model training phase.
- 2 attacks: HopSkipJumpAttack and Zeroth Order Optimization.
- 1 packet validator: specialized to the selected data sets.

This construction was motivated by our aims to obtain experimentally interesting variance, preference to use existing advanced learning algorithms and attacks, and the need to maintain software compatibility between system components. We relied heavily on Adversarial Robustness Toolkit (ART) [25] to reach those objectives. We also implemented 1.5K lines of custom Python code, mainly to create the packet validator and to coordinate experiments. The implementation enables to measure attack success rate for different experiment configurations, and to reproduce all evaluation results presented in this paper. The source code and data are available online, with instructions how to reproduce and reuse our experiments [29][1].

4.1 Benchmark Data Sets

UNSW-NB15 and IoT-23 data sets contain traditional and IoT network traffic patterns respectively. They represent modern network flows, include multiple attack classes, and are pre-labelled for supervised learning. Their experimentally relevant properties are summarized in Table 2.

[1] For long-term archival, the repository is preserved by Software Heritage with snapshot ID swh:1:rev:609c59c320f732c744d0d3a05f55041e4cdc878d.

Table 2. Overview of data sets used for evaluation.

Data set properties	IoT-23	UNSW NB15	Experimental setup	IoT-23	UNSW NB15
Network kind	IoT	classic	Record count	10,000	10,000
Attacks	20	9	Features/binarized	11/18	24/32
Features	23	49	Mutable/immutable	8/10	22/10
Flow records	300M	2M	TCP/UDP %	52/47	52/33
Capture timeframe	2018–19	2015	Other protocols %	<1	15
Publication year	2020	2015	Class distribution	50/50	50/50

Aposemat IoT-23 [10] is a large network traffic capture of Internet of Things devices. It was generated in a controlled environment on real hardware, including voice assistants, smart LED lamps, and smart door locks. The initial records, in pcap format, were processed – manually and using network analysis tools – into 23 features and binary class labels. It contains twenty malware captures: Mirai, Torii, DDoS, etc., and three captures of benign traffic.

UNSW-NB15 [21–24,30] is a benchmark data set of traditional network traffic. It contains a mix of real normal traffic flows and contemporary synthesized attack activities. The traffic flows were generated using intrusion detection and network flow analysis tools to obtain 49 features and binary class labels. It has nine attack classes: Fuzzers, DoS, Backdoor, etc., and one benign class.

We applied similar preprocessing steps to both data sets[2]. We removed redundant attributes, one-hot encoded nominal attributes, and binarized class labels. We did not normalize data until runtime. This delay allowed us to inspect adversarial examples in their original numeric ranges, which improved interpretability. We constructed 10,000-record data sets of equal class distribution by random sampling without replacement. IoT-23 experiment data was sampled from CTU malware captures 1–1, 8–1, and 34–1, which contain Hakai, Mirai and Hide-and-Seek attack flows. UNSW-NB15 experiment data was sampled from the pre-assembled 175K-record training set.

4.2 Classifiers and Defenses

We selected two supervised learners to construct classifiers: XGBoost [36], a tree-based ensemble learner, and Keras [14] Deep Neural Network (DNN). Several factors supported their selection as attack target classifiers. Both are advanced learning algorithms, offer high degree of configurability, and are widely used in practice. Their uses in NIDS have been researched previously [8,12,17,19]. Initial experiments showed they obtained high accuracy on the selected data sets. Lastly, both classifiers were compatible with training phase adversarial defenses.

We trained XGBoost for 20 boost-rounds using a softmax objective and exact tree method for both data sets. Because IoT-23 data set has fewer features than

[2] Detailed preprocessing steps are in the experiment repository data/readme.

UNSW-NB15, and neural network training is resource-intensive, we constructed different DNNs for the two data sets. We set 1 hidden layer for IoT-23 and 5 hidden layers for UNSW-NB15. All hidden layers had 60 neurons, used standard ReLU activation function, and lastly softmax to determine output[3].

To train robust models, we applied RobustTrees [4] defense to XGBoost. It is an algorithmic approach, that finds the best split, measured by optimized performance under the worst-case perturbations. For DNN, we used Adversarial Training [33]. It works by inserting adversarially generated records into training data to produce a robust model. The intuition is that seeing adversarial training examples improves model's ability to detect malicious examples during prediction. A defense can be enabled or disabled during an experiment. We use 🛡 symbol to identify robust models.

4.3 Adversarial Attacks

We selected two black-box evasion attacks for evaluation: HopSkipJumpAttack and Zeroth Order Optimization. The attacks have multiple differences that make them interesting to compare in constrained domains. In addition, it was necessary to identify attacks compatible with the selected classifiers. We obtained implementations of these attacks from ART [25].

Zeroth order optimization (ZOO) [6] is a score-based attack. A score-based attack aims to optimize function f based on its values $f(x)$ and assumes model queries provide class label probabilities [5]. ZOO works by first querying the target model. The response contains class confidence scores for the input. Then, it uses the response scores to estimate gradients of the target model and uses that information to generate adversarial examples. ZOO allows universal feature perturbations and requires a high number of model queries [15]. ZOO was originally introduced to target DNN models trained on images [6].

HopSkipJumpAttack [5] works by estimating gradient direction and requires only binary information available at the decision boundary. Intuitively it works by performing a binary search to identify a decision boundary. Then it estimates a gradient at the boundary and uses geometric progression to generate adversarial examples. It uses a decision-based scoring model, where model query responses contain only class labels. Compared to a score-based approach, a decision-based scoring model is more difficult and realistic, because it assumes less available information. HopSkipJumpAttack is also query-efficient: it requires fewer model queries than other decision-based adversarial attacks. These features make it an interesting evaluation case in NIDS, although it was introduced for image data.

A constrained domain requires to prevent perturbations of immutable attributes of adversarially generated examples. This is one – but not the only – example of a constraint imposed by the domain. ART supports this requirement with a technique called *masking*. It requires to provide the indices of immutable

[3] The exact classifier training configurations are in the config/ directory, with some parameters computed dynamically in xgb.py and dnn.py.

attributes as an attack parameter. Those attributes cannot be operated on and will retain their original values. HopSkipJumpAttack supports masking natively, but ZOO does not. For ZOO, we used a mechanism of *pseudo-masking*, which works as follows. (i) Identify immutable attribute indices before attack. (ii) Allow perturbations to be applied freely over the feature space. (iii) Restore immutable attributes to their original values after the attack. Pseudo-masking achieves the same outcome as native masking. However, it is suboptimal because adversarial example generation cannot leverage information about attribute immutability.

4.4 Validator Implementation

The packet validator was developed concretely for the selected data sets. It was necessary to customize the validator by data set because, after preprocessing, the remaining data set features did not intersect. Our development shows the exact implementation details[4]. The approach is generalizable and can be extended to other data sets and network protocols.

The validator is a custom module that takes as input an adversarial example. Next it determines the example's network protocol and checks it against applicable rules. A valid adversarial example must satisfy all rules. The default assumption is permissive: if an example cannot be shown to violate a rule, the check will pass. For additional insight, we collect statistics of violated rules during experiments. This allows to identify, for invalid adversarial examples, which attributes are involved and at what frequency those violations occur.

Implementing the validation rules was not strictly straightforward. Real captures include measurement noise and other anomalies may arise from synthetic network captures. A demonstrative issue occurred with TCP/IP packet start and end time measurements. A precise rule states that relation starttime + duration + ϵ = endtime should hold. In practice this level of precision was unattainable, because different measurement precision was applied to these attributes. We resolved the issues by relaxing the rules. For example, the time-relation can be expressed without the middle operands as starttime $\leq$ endtime.

5 Evaluation Results

We used our approach and its implementation to evaluate the current state-of-the-art in adversarial attacks and defenses in NIDS. The results show adversarial success in 34 of 48 evaluated system configurations of restricted model queries, and in 7 out of 8 cases of restricted input size.

5.1 Experiment and Environment Configuration

The implementation enables constructing and measuring various configurations within the experimental setup. In each experiment, we applied k-folds cross validation, where $k = 5$. We trained a classifier on $k - 1$ subsets, and an applied an

[4] See validator.py for Validator class and Protocol base class.

Table 3. Classifier training results, $\mu \pm \sigma$.

Model	IoT-23			UNSW-NB15		
	F-score	Accuracy	Recall	F-score	Accuracy	Recall
DNN	.98 ± .0	0.98 ± .0	.95 ± .00	.87 ± .02	.87 ± .02	.89 ± .06
DNN-🛡	.97 ± .0	0.98 ± .0	.95 ± .01	.79 ± .01	.77 ± .01	.89 ± .01
XGB	.98 ± .0	0.98 ± .0	.96 ± .01	.94 ± .01	.94 ± .01	.95 ± .01
XGB-🛡	.98 ± .0	0.98 ± .0	.95 ± .01	.82 ± .01	.82 ± .01	.85 ± .01

adversarial attack to the validation set. For each fold, we performed model training and attack from start, i.e., each fold represents an independent experiment instance. We report results as an arithmetic mean of k-folds.

A successful adversarial attack requires following three conditions to hold.

(1) The original record must be correctly classified by the target model. Since only true labels can be misclassified, model accuracy controls the upper bound of adversarial success rate.
(2) The adversarial record must produce a misclassification. For a binary problem this means the class label is reversed.
(3) The adversarial record must pass structural correctness check performed by the packet validator. Only evasive records need to be validated, since both conditions must hold.

We give two breakdowns of this criteria in the results. "Evasions" is the fractions of records that satisfy the first two conditions. "Valid" is the fraction of records that satisfy all three conditions. Adversarial attack success rate is the fraction of adversarially generated records that satisfy all three criteria, out of total records.

We performed experiments on a Linux Ubuntu v18.04 Bionic machine, with x86-64 architecture, 8 cores, and 32 GB of RAM. The experiments do not require specialized hardware but take over 24h to complete on the specified machine. In our source code repository [29], we include a detailed list of software dependencies and a referential result on which the values reported in this paper are based[5].

5.2 Observations

Model training results are presented in Table 3. It includes recommended performance metrics for ML-based NIDS [1, p. 21]. Model accuracy is high for all IoT-23 models. Adversarial defenses impact is more detectable in UNSW-NB15 models, where metrics vary noticeably between robust and non-robust models. This is expected because the data sets differ in feature counts and model complexity.

Measuring Model Query Restriction. In Table 4, using a validation set of 2,000 records, we applied adversarial attacks with restricted model queries. This

[5] See requirements.txt for dependencies and resut/ for captured results.

Table 4. Adversarial attack success rate for 48 attack configurations, represented as *fractions*. "Valid" represents the fraction of evasive records that also pass validation. Iterations 10 (HopSkipJumpAttack) and 80 (ZOO) are the attack's default iterations, i.e., baseline cases.

Model/Iterations	HopSkipJumpAttack						Zeroth Order Optimization					
	Evasions			Valid			Evasions			Valid		
	2	5	10	2	5	10	2	5	80	2	5	80
IoT-23												
DNN	.34	.27	.31	0	0	.01	0	0	0	0	0	0
DNN-🛡	0	0	0	0	0	0	0	0	0	0	0	0
XGB	.43	.39	.41	.06	.07	.18	.47	.49	.49	.05	.05	.04
XGB-🛡	.38	.38	.38	.01	.01	.03	.03	.07	.07	.03	.06	.07
UNSW-NB15												
DNN	.79	.68	.81	.41	.39	.42	.28	.36	.29	.25	.30	.24
DNN-🛡	.02	.11	.07	.02	.11	.07	0	0	0	0	0	0
XGB	.93	.92	.91	.47	.46	.47	.50	.69	.78	.49	.65	.69
XGB-🛡	.64	.65	.65	.38	.38	.38	.09	.31	.32	.09	.30	.31

Table 5. UNSW-NB15. Adversarial success rate by network transmission protocol, for baseline iterations. Column Benign–Malicious, B–M, shows class labels distribution of evasive and valid records.

Model/Protocol	Evasions			Valid			B–M
	TCP	UDP	other	TCP	UDP	other	
HopSkipJumpAttack							
DNN	.79	.85	.81	.78	.02	.03	27–73
DNN-🛡	.14	0	0	.14	0	0	0–100
XGB	.91	.94	.88	.89	.02	.01	30–70
XGB-🛡	.75	.43	.78	.73	0	0	17–83
Zeroth Order Optimization							
DNN	.35	.23	.22	.34	.13	.14	52–48
DNN-🛡	0	0	0	0	0	0	–
XGB	.89	.70	.55	.88	.50	.43	34–66
XGB-🛡	.54	.11	.01	.53	.11	.01	24–76

Table 6. UNSW-NB15. Adversarial attack average success rate for limited input size of 50 records, for average of 3 rounds of random sampling.

Model	HopSkipJumpAttack			Zeroth Order Optimization		
	Evasions	Valid	B–M	Evasions	Valid	B–M
DNN	.81	.48	28–72	.30	.27	46–54
DNN-🛡	.01	.01	0–100	0	0	–
XGB	.95	.49	34–66	.79	.70	38–62
XGB-🛡	.68	.38	20–80	.34	.34	25–75

Table 7. Observed validator rule violations. Frequency is relative to attack and protocol. It gives fraction of rejections based on the specified reason. Inspecting these reasons provides insight of modified attributes and why they cause failure.

IoT-23			UNSW-NB15		
Protocol	Rule	Freq	Protocol	Rule	Freq
HopSkipJumpAttack					
TCP	S0 packets or bytes $\neq 0$	.982	TCP	invalid swin/dwin	.743
	ori packets $>$ bytes	.014		dbytes $\neq 0$ and dur $= 0$	.257
	ori packets, but bytes=0	.003	UDP	nonzero tcp fields	1.0
	resp packets $>$ bytes	.001	other	nonzero tcp fields	1.0
Zeroth Order Optimization					
TCP	ori packets $>$ bytes	1.0	TCP	dbytes $\neq 0$ and dur $= 0$	1.0
			UDP	nonzero tcp fields	1.0
			other	nonzero tcp fields	1.0

restriction is captured by maximum limit of attack *iterations*. It represents a scenario where repeated model queries are limited by network security mechanisms. Intuitively, decreasing model queries should decrease attack success rate, and this experiment aims to measure that impact. We observe the attacker's success rate is slightly lower at 2 and 5 iterations. However, the adversary never ceases to be successful under these restrictions. The results also confirm the evaluated adversarial defenses are effective in reducing attacker's success rate. Increasing query limit is more impactful for ZOO attack. It has less impact on HopSkipJumpAttack, which is by design query efficient. Overall, HopSkipJumpAttack shows more frequent success and higher attack success rate in most evaluated cases.

Now, further referring to Table 4, we note that successful adversarial attacks on IoT-23 models were more difficult to achieve. Adversary was successful against XGBoost models and non-robust DNN using HopSkipJumpAttack, i.e., 5 out of 8 cases. We hypothesize this is due to model simplicity and low number of mutable attributes in the IoT-23 data set. This means the attack surface is smaller and makes attack success more challenging to achieve. Following this discovery, we applied the detailed analyses only to UNSW-NB15 models.

In Table 5, the B–M (Benign–Malicious) column shows that adversary successfully evades classification for both class labels. Cases marked benign are initially malicious, and malicious labels are initially benign examples. After misclassification, their labels have changed. The benign case, where a record was initially malicious, is more harmful as it can compromise system integrity. The table shows the adversary's success rate at crafting the more harmful packets is lower than the opposite misclassification. However, the adversary is successful at causing misclassification in both directions in all experiment scenarios, except against DNN-🛡. Comparing the attacks, ZOO yields slightly higher success rate at disguising malicious packets as benign.

Table 5 also gives a protocol-specific breakdown of the results. It shows higher adversarial success rate for TCP records. This is expected because initial protocol distribution is higher for TCP packets in the data set. In Table 7, we see UDP and other protocol records are modified illegally, which causes validation failure. The TCP examples are mostly modified in acceptable ways and pass the packet validator check.

Measuring Restricted Input-Size. In this experiment, adversarial attacks were applied on a subset of records sampled from the validation set. This scenario represents a case where data size must remain small to avoid detection, but model queries are not restricted. We performed 3 rounds of sampling, to obtain a random subset of 50 records, and report the arithmetic mean success rate of those 3 rounds.

In Table 6, we observe the attacker's success rate remains comparable to the baseline. Therefore, it remains possible for attacker to be successful even if attacked can only afford a few different input perturbations. Expressed differently, having in place a network defense mechanism that blocks large quantities of packets is, by these results, insufficient to prevent an adversary from crafting valid adversarial inputs.

Discussion. Overall, we can observe the following. AML, specifically black-box evasion attacks, continues to be a threat to NIDS. In other words, despite constraints—both what makes a valid adversarial packet, and limited number of iterations and query size—attackers can evade an existing model. Furthermore, while methods for improving the robustness of models have some success, they cannot fully prevent successful evasions. The results are dependent upon data set, model, attack method, and even network protocol, which suggests that more evaluation is needed. In that regard, our validator-based approach provides a mechanism to easily experiment with attacks developed for unconstrained spaces.

6 Related Work

The existence of adversarial threat in NIDS is covered abundantly—by discussing its relevance and potential [12,15,16], and by empirically measuring its impact [2,11,17,31]—and therefore is a foundational assumption for this discussion.

Several countermeasures have been suggested to mitigate adversarial threats in NIDS. In addition to the defenses presented in Sect. 5, multiple algorithmic approaches have been proposed recently. These approaches leverage, e.g., hierarchical learning [17] and feature grouping (FGMD) [13]. An alternative defense strategy is presented in Kitsune, a NIDS with built-in adversarial robustness; it can learn attacks in online manner and without supervision [19]. FGMD and Kitsune are interesting because they are designed for network devices with limited computational resources. New approaches to crafting adversarially generated examples have also been proposed. These techniques include, e.g., adversarial sketches [31], evolutionary computation [2], and model extraction with saliency maps [27]. The perpetual game of chase between adversarial attacks and defenses is observable in the fact that some attack evaluations establish their relevance by compromising Kitsune [11,27]. As stated throughout, the distinguishing aspect of our work is in the ability to apply attacks from unconstrained domains to a constrained domain.

A more directly comparable source of related studies focuses on evaluation of existing techniques and threat scenarios. As groundwork, Sheatsley et al. [31] demonstrated that adversarial success was possible in constrained domains with only 5 mutable features, and that the constrained nature of networks is insufficient to mitigate the threat. An evaluation of classification algorithms showed that tree-based NIDS were notably vulnerable to adversarial threats [2]. Other evaluation studies have focused on, e.g., neural network-based NIDS [11], evaluations of white-box attacks [35], and black-box attacks [38]. However, Apruzzese et al. [3] boldly posit that existing evaluations are mostly unrealistic—due to assumptions of complete knowledge and freedom to interact with the target system—and suggest an alternative evaluation model for performing realistic evaluations. The main differentiating factor between these works and ours is the generalizable nature of our evaluation system. Our approach can leverage results from other domains and evaluate realistic black-box attacks, while incorporating the necessary steps to enforce validity of generated adversarial examples.

7 Conclusions

We have focused on the problem of evaluating AML – specifically, evasion attacks – in the context of network intrusion detection. As opposed to the approaches that try to develop new attacks for a constrained domain, we have presented a novel approach that can enable existing techniques and software systems for an unconstrained domain to be applied and evaluated for our target domain. The key idea here is to add a post-hoc validator that can prune away adversarial records that do not meet the constraints of the domain. We demonstrated the practicality and feasibility of our design through an implementation and concrete experimental results. These findings experimentally substantiate that constrained environments continue to be susceptible to adversarial attacks. The results also demonstrate efficiency of our rule-based validator in performing this critical task.

While our work and implementation experimentally demonstrate the usefulness of this post-hoc approach, an interesting question remains about incorporating mechanisms of constrain-validation *during* the adversarial search phase. Having insight of constraints could guide the search, and a sophisticated attacker could leverage the validator feedback to improve success rate at crafting evasive and valid adversarial examples. We leave exploration of these open questions for future work.

Acknowledgments. The authors thank the developers of Adversarial Robustness Toolkit [25], particularly Beat Buesser for helpful discussion, and for providing and maintaining software that helped to facilitate our experiments.

References

1. Ahmad, Z., Shahid Khan, A., Wai Shiang, C., Abdullah, J., Ahmad, F.: Network intrusion detection system: a systematic study of machine learning and deep learning approaches. Trans. Emerg. Telecommun. Technol. **32**(1), e4150 (2021). https://doi.org/10.1002/ett.4150
2. Alhajjar, E., Maxwell, P., Bastian, N.: Adversarial machine learning in network intrusion detection systems. Expert Syst. Appl. **186**, 115782 (2021). https://doi.org/10.1016/j.eswa.2021.115782
3. Apruzzese, G., Andreolini, M., Ferretti, L., Marchetti, M., Colajanni, M.: Modeling realistic adversarial attacks against network intrusion detection systems. Digit. Threats: Res. Pract. (DTRAP) **3**(3), 1–19 (2022). https://doi.org/10.1145/3469659
4. Chen, H., Zhang, H., Boning, D., Hsieh, C.J.: Robust decision trees against adversarial examples. In: International Conference on Machine Learning, pp. 1122–1131. PMLR (2019). http://proceedings.mlr.press/v97/chen19m/chen19m.pdf
5. Chen, J., Jordan, M.I., Wainwright, M.J.: HopSkipJumpAttack: a query-efficient decision-based adversarial attack. In: 2020 IEEE symposium on security and privacy, pp. 1277–1294. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00045
6. Chen, P.Y., Zhang, H., Sharma, Y., Yi, J., Hsieh, C.J.: ZOO: zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec 2017), pp. 15–26 (2017). https://doi.org/10.1145/3128572.3140448
7. CleverHans Lab: CleverHans v4.0.0 (2021). https://github.com/cleverhans-lab/cleverhans
8. Dhaliwal, S.S., Nahid, A.A., Abbas, R.: Effective intrusion detection system using XGBoost. Information **9**(7), 149 (2018). https://doi.org/10.3390/info9070149
9. Ding, G., et al.: AdverTorch v0.2 (2019). https://github.com/BorealisAI/advertorch
10. Garcia, S., Parmisano, A., Erquiaga, M.J.: IoT-23: a labeled dataset with malicious and benign IoT network traffic (2020). https://doi.org/10.5281/zenodo.4743746
11. Hashemi, M.J., Cusack, G., Keller, E.: Towards evaluation of NIDSs in adversarial setting. In: Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks, Big-DAMA 2019, pp. 14–21. ACM (2019).https://doi.org/10.1145/3359992.3366642
12. He, K., Kim, D.D., Asghar, M.R.: Adversarial machine learning for network intrusion detection systems: a comprehensive survey. IEEE Commun. Surv. Tutor. **25**(1), 538–566 (2023). https://doi.org/10.1109/COMST.2022.3233793
13. Jiang, H., Lin, J., Kang, H.: FGMD: a robust detector against adversarial attacks in the IoT network. Future Gener. Comput. Syst. **132**, 194–210 (2022). https://doi.org/10.1016/j.future.2022.02.019
14. Keras team: Keras: Deep Learning for humans v2.11.0 (2022). https://github.com/keras-team/keras
15. Martins, N., Cruz, J.M., Cruz, T., Abreu, P.H.: Adversarial machine learning applied to intrusion and malware scenarios: a systematic review. IEEE Access **8**, 35403–35419 (2020). https://doi.org/10.1109/ACCESS.2020.2974752
16. McCarthy, A., Ghadafi, E., Andriotis, P., Legg, P.: Functionality-preserving adversarial machine learning for robust classification in cybersecurity and intrusion detection domains: A survey. J. Cybersecur. Priv. **2**(1), 154–190 (2022). https://doi.org/10.3390/jcp2010010

17. McCarthy, A., Ghadafi, E., Andriotis, P., Legg, P.: Defending against adversarial machine learning attacks using hierarchical learning: a case study on network traffic attack classification. J. Inf. Secur. Appl. **72**, 103398 (2023). https://doi.org/10.1016/j.jisa.2022.103398
18. McFowland, E., Speakman, S., Neill, D.B.: Fast generalized subset scan for anomalous pattern detection. J. Mach. Learn. Res. **14**(1), 1533–1561 (2013). https://www.jmlr.org/papers/volume14/mcfowland13a/mcfowland13a.pdf
19. Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A.: Kitsune: an ensemble of autoencoders for online network intrusion detection. In: Network and Distributed Systems Security (NDSS) Symposium 2018, pp. 1–15. NDSS (2018).https://doi.org/10.14722/ndss.2018.23204
20. Mitchell, T.M.: Machine Learning, 1st edn. McGraw Hill (1997)
21. Moustafa, N., Creech, G., Slay, J.: Big data analytics for intrusion detection system: statistical decision-making using finite Dirichlet mixture models. In: Palomares Carrascosa, I., Kalutarage, H.K., Huang, Y. (eds.) Data Analytics and Decision Support for Cybersecurity. DA, pp. 127–156. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59439-2_5
22. Moustafa, N., Slay, J.: UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: 2015 Military Communications and Information Systems Conference (MilCIS), pp. 1–6. IEEE (2015). https://doi.org/10.1109/MilCIS.2015.7348942
23. Moustafa, N., Slay, J.: The evaluation of network anomaly detection systems: statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. Inf. Secur. J.: Glob. Perspect. **25**(1–3), 18–31 (2016). https://doi.org/10.1080/19393555.2015.1125974
24. Moustafa, N., Slay, J., Creech, G.: Novel geometric area analysis technique for anomaly detection using trapezoidal area estimation on large-scale networks. IEEE Trans. Big Data **5**(4), 481–494 (2017). https://doi.org/10.1109/TBDATA.2017.2715166
25. Nicolae, M-I., et al.: Adversarial robustness toolbox v1.12.1 (2022). https://github.com/Trusted-AI/adversarial-robustness-toolbox
26. Oprea, A., Vassilev, A.: Adversarial machine learning - a taxonomy and terminology of attacks and mitigations. In: NIST Artificial Intelligence (AI) NIST AI, pp. 1–74 (03 2023). https://doi.org/10.6028/NIST.AI.100-2e2023.ipd
27. Qiu, H., Dong, T., Zhang, T., Lu, J., Memmi, G., Qiu, M.: Adversarial attacks against network intrusion detection in IoT systems. IEEE Internet Things J. **8**(13), 10327–10335 (2021). https://doi.org/10.1109/JIOT.2020.3048038
28. Rauber, J., et al.: Foolbox native v3.3.3 (2022). https://github.com/bethgelab/foolbox
29. Rusch, N., et al.: Evaluating AML threats in NIDS v. 1.0 (2023). https://github.com/aucad/aml-networks
30. Sarhan, M., Layeghy, S., Moustafa, N., Portmann, M.: NetFlow datasets for machine learning-based network intrusion detection systems. In: Deze, Z., Huang, H., Hou, R., Rho, S., Chilamkurti, N. (eds.) BDTA/WiCON -2020. LNICST, vol. 371, pp. 117–135. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72802-1_9
31. Sheatsley, R., Papernot, N., Weisman, M.J., Verma, G., McDaniel, P.: Adversarial examples for network intrusion detection systems. J. Comput. Secur. **30**(5), 727–752 (2022). https://doi.org/10.3233/JCS-210094
32. Tan, P.N., Steinbach, M., Karpatne, A., Kumar, V.: Introduction to Data Mining, 2nd edn. Pearson (2019)

33. Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., McDaniel, P.: Ensemble adversarial training: attacks and defenses. arXiv preprint arXiv:1705.07204 (2017). https://doi.org/10.48550/arXiv.1705.07204
34. Wang, X., Li, J., Kuang, X., Tan, Y.A., Li, J.: The security of machine learning in an adversarial setting: a survey. J. Parallel Distrib. Comput. **130**, 12–23 (2019). https://doi.org/10.1016/j.jpdc.2019.03.003
35. Wang, Z.: Deep learning-based intrusion detection with adversaries. IEEE Access **6**, 38367–38384 (2018). https://doi.org/10.1109/ACCESS.2018.2854599
36. XGBoost developers: XGBoost v. 0.72 Documentation. https://xgboost.readthedocs.io/en/release_0.72/. Accessed 27 Nov 2022
37. Xu, W., Evans, D., Qi, Y.: Feature squeezing: detecting adversarial examples in deep neural networks. In: Proceedings 2018 Network and Distributed System Security Symposium, pp. 1–15. Internet Society (2018). https://doi.org/10.14722/ndss.2018.23198
38. Yang, K., Liu, J., Zhang, C., Fang, Y.: Adversarial examples against the deep learning based network intrusion detection systems. In: MILCOM 2018–2018 IEEE Military Communications Conference (MILCOM), pp. 559–564. IEEE (2018). https://doi.org/10.1109/MILCOM.2018.8599759
39. Yun, S., Han, D., Oh, S.J., Chun, S., Choe, J., Yoo, Y.: CutMix: regularization strategy to train strong classifiers with localizable features. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 6022–6031 (2019). https://doi.org/10.1109/ICCV.2019.00612