Adversarial Examples for Network Intrusion Detection Systems

Ryan Sheatsley [a,*], Nicolas Papernot [b,**], Michael J. Weisman [c], Gunjan Verma [c] and Patrick McDaniel [a]

[a] *Department of Computer Science and Engineering, The Pennsylvania State University, PA, United States*
E-mails: sheatsley@psu.edu, mcdaniel@cse.psu.edu
[b] *Department of Computer Science, University of Toronto, Ontario, Canada*
E-mail: nicolas.papernot@utoronto.ca
[c] *United States Army Research Laboratory, MD, United States*
E-mails: michael.j.weisman2.civ@mail.mil, gunjan.verma.civ@mail.mil

Abstract.
Machine learning-based network intrusion detection systems have demonstrated state-of-the-art accuracy in flagging malicious traffic. However, machine learning has been shown to be vulnerable to adversarial examples, particularly in domains such as image recognition. In many threat models, the adversary exploits the unconstrained nature of images–the adversary is free to select some arbitrary amount of pixels to perturb. However, it is not clear how these attacks translate to domains such as network intrusion detection as they contain *domain constraints*, which limit which and how features can be modified by the adversary. In this paper, we explore whether the constrained nature of networks offers additional robustness against adversarial examples versus the unconstrained nature of images. We do this by creating two algorithms: (1) the ADAPATIVE-JSMA, an augmented version of the popular JSMA which obeys domain constraints, and (2) the HISTOGRAM SKETCH GENERATION which generates *adversarial sketches*: targeted universal perturbation vectors that encode feature saliency within the envelope of domain constraints. To assess how these algorithms perform, we evaluate them in a constrained network intrusion detection setting and an unconstrained image recognition setting. The results show that our approaches generate misclassification rates in network intrusion detection applications that were comparable to those of image recognition applications (greater than 95%). Our investigation shows that the constrained attack surface exposed by network intrusion detection systems is still sufficiently large to craft successful adversarial examples–and thus, network constraints do not appear to add robustness against adversarial examples. Indeed, even if a defender constrains an adversary to as little as five random features, generating adversarial examples is still possible.

1. Introduction

Network intrusion detection systems (NIDS) have traditionally manifested as rule-based systems, such as Bro [42] and Snort [2]. However, with the rise in popularity of machine learning (ML), we are observing that machine-learning-based network intrusion detection systems are outperforming their rule-based counterparts [8, 41, 43, 50]. While the performance of ML-based NIDS is compelling for

[*] Corresponding author. E-mail: sheatsley@psu.edu.
[**] Work done while the author was at the Pennsylvania State University

widespread adoption and deployment, they are not void of their own limitations: when an adversary is introduced machine learning and the myriad of domains they serve often become vulnerable.

The field of adversarial machine learning (AML) explores the impact of *adversarial examples*: inputs to machine learning models that an attacker has designed to cause the model to make a mistake, e.g., misclassify [11]. Within the scope of this research, there have been studies that target particular inputs through a variety of threat models, with algorithms such as the JACOBIAN-SALIENCY MAP APPROACH [36], CARLINI-WAGNER [5], and PROJECTED GRADIENT DESCENT [27], among others. A different class of attacks (exposing potentially more serious vulnerability) compute *universal adversarial perturbations* [12, 30]. These perturbations can be precomputed and quickly applied at runtime to arbitrary inputs while maintaining adversarial goals (e.g., misclassification of arbitrary inputs to a class chosen by the adversary). Investigations have shown that no domain (thus far) is immune to this phenomenon; the scope of adversarial examples has been expansive, reaching into image processing [5, 12, 29, 36], malware detection [13, 22], text [9, 20], and even speech recognition [6]. The existence of adversarial examples presents a compelling barrier for sensitive domains that use machine learning.

To date, many published works in adversarial machine learning have focused on *unconstrained* domains, that is, domains in which the adversary may arbitrarily select and perturb features by some parameterized cost (often measured through some l_p-norm). Implicit to this threat model is that such freedom overestimates the capabilities of an adversary in constrained domains, such as network intrusion detection, where they are often bound by the semantics of features (and realistically capable of only controlling some subset of features). Thus, we expect such algorithms used to generate adversarial examples would be less effective in constrained domains. In this work, we consider network intrusion detection as our exemplary constrained domain, using two popular network intrusion detection datasets: the NSL-KDD [47] and UNSW-NB15 [31]. Here, *constraints* are defined by the following three characteristics: the values within a feature may be fixed (binary vs continuous), the values of different features may be correlated (TCP flags in packets and TCP as the transport protocol), and some features may not be controllable by an adversary (round-trip times).

Furthermore, while there has been research that has focused on constrained *adversaries*[13, 45], who are bound in their capabilities (for example, how many total features can be perturbed), we differentiate in that we consider constrained *domains*. Specifically, these constraints describe the kinds of inputs that are *permissible* in a domain (e.g., network packets that do not obey the TCP/IP protocol would not be permissible). In this work, we measure the efficacy of adversaries when subjected to the union of both adversary *and* domain constraints in networks, unlike previous work.

In this paper, we hypothesize that ML-based network intrusion detection systems are more robust to adversarial examples from two threat models: (1) from traditional adversarial algorithms, and (2) from universal adversarial perturbations. First, introduce the notion of *primary features*, which constrain the values other features can have. We design an algorithm to extract constraints from network datasets. Then, we develop an augmented algorithm, the ADAPTIVE JSMA (AJSMA), which not only improves on the state-of-the-art in attack algorithms, but also constructs adversarial examples that obey the extracted constraints. Next, we design a second algorithm, the HISTOGRAM SKETCH GENERATION (HSG), the first attack to compute *adversarial sketches*: universal perturbations used to craft adversarial examples en masse, while also complying with the extracted constraints. With both algorithms, we measure the success rate of crafting adversarial examples by attacking models directly in both network (i.e., constrained) and image (i.e., unconstrained) domains, and find that we were comparably successful in both domains (misclassification rates greater than 95%). Furthermore, we perform a series of grey-box attacks where we use adversarial examples crafted from a source model to attack target models with different learning techniques and trained on disjoint subsets of a dataset. Our results demonstrate that adversarial examples also *transfer* in the constrained network domain (reaching up to 93% for the AJSMA and 100% for the HSG). Finally, we show that even if an adversary maintains attack behavior *and* cannot arbitrarily control certain features *and* must obey the TCP/IP protocol, there is still a surprising amount of exploitable attack surface to craft adversarial examples. We make four contributions:

(1) We introduce domain constraints for network intrusion detection systems, and design an algorithm that is able to extract them systematically. Extracting constraints codifies the space of permissible adversarial examples.
(2) We introduce two algorithms: (1) The ADAPTIVE JSMA, which produces adversarial examples that obey network constraints, and (2) the HISTOGRAM SKETCH GENERATION, which produces adversarial sketches: universal adversarial perturbations that obey networks constraints.
(3) We perform experiments where we impose an extreme amount of adversary constraints, to the point where an adversary can only control five random features, and show that adversarial examples can still be crafted with a $\sim 50\%$ success rate while also complying with domain constraints.
(4) We demonstrate promising results for both algorithms, reaching greater than 95% misclassification rates across the datasets used in our experiments. This suggests that ML-based NIDS are comparably as vulnerable as their unconstrained counterparts, such as image recognition systems.

2. Background

Adversarial machine learning research for image recognition systems has been broad. Since the observations of Biggio et al. and Szegedy et al. in deep neural networks [3, 46] to the robust attacks from Kurakin et al. and Sharif et al. [23, 40], adversarial examples have matured from, "an intriguing property" to a tangible threat.

The first generation of attacks were formed in the context of "white-box" attacks [5, 12, 36]. Under this threat model, adversarial examples are crafted using information (e.g., model parameters) directly from the model under attack. This represents a worst-case scenario, analogous to an insider threat, since such information would not be easily accessible in most practical contexts. Naturally, this motivates the question: Can an adversary successfully attack a model, *without having direct access to its parameters?* Papernot et al. and Tramèr et al. investigated this question by leveraging *transferability*: an adversarial example crafted from one model will often be an adversarial example for a different model, even if they are using different training data and/or learning techniques [37, 49]. Through this "black-box," threat model, an adversary trains a *surrogate* model by using inputs to generate output labels from the victim model, called an *oracle*. Afterwards, the surrogate model is used to craft adversarial examples which are then (with high probability) "transferred" to the victim model.

Concurrently, others have investigated what limitations (if any) exist for adversarial examples. Kurakin et al. and Brown et al. explored how adversarial examples can be applied directly to the physical domain, introducing techniques that produce adversarial examples robust to physical distortions, such as rotation, scale, and other transformations [4, 23]. Moreover, Goodfellow et al. and Moosavi-Dezfooli et al. analyzed *universal adversarial perturbations*: single perturbation vectors used to quickly craft adversarial examples from many inputs not known in advance [12, 30]. These universal perturbations are especially concerning as they enable adversaries to take computation offline and amortize computational costs over many inputs compared to traditional, online attacks.

For this work, we modify an existing attack, the *Jacobian-based Saliency Map Approach*, introduced by Papernot et al. [36]. The JSMA produces an adversarial example by greedily applying perturbations to the most salient features in an input. The algorithm terminates when either the input is successfully misclassified or the specified l_0 distance[1]is reached. The JSMA greedily selects features to perturb by constructing *saliency maps*, which encode the influence features have over misclassifying a particular input. This l_0 minimization makes the JSMA an attractive candidate for benchmarking the security of ML-based NIDS, as discussed in §3. However, the JSMA is not necessary for crafting adversarial examples against ML-based NIDS. Different attack algorithms can be used, with some adjustments, which we defer to future work.

2.1. Adversarial Examples for NIDS

Unlike traditional image-based evaluations in adversarial machine learning, crafting adversarial examples for network intrusion detection systems is a necessarily different process. For one, the measurements used for bounding adversarial perturbations are simply inappropriate: traditional l_2 or l_∞ norm budgets [5, 12, 46] have been used as surrogate for human perception. However, networks are inherently a non-visual domain, and thus, bounding adversaries based on human perception is not suitable.

Moreover, common assumptions in images allow adversaries to perturb features *arbitrarily and independently*—that is, any adversarial example is equally permissible in the domain. However, in networks, this assumption does not hold: a single network flow cannot connect to multiple services within the same packet, cannot set TCP flags on a UDP flow, cannot have negative round-trip times, among other constraints that, if ignored, adversarial crafting algorithms will produce. Thus, the threat model in networks is unique in that there are *domain constraints* that adversaries must obey for an adversarial example to be attack that represents a realizable traffic flow.

Finally, while there are some works that investigated the efficacy of adversaries in domains such as network intrusion detection, we find that the domain constraints: (1) are ignored [38, 51], or (2) are explicitly avoided when applying perturbations [26]. Clearly, ignoring domain constraints in attack algorithms could produce adversarial examples that cannot be traced back to any traffic flow. Moreover, identifying features that are unconstrained reduces to the same approaches for attacking image data, in that adversaries are also free to perturb arbitrarily and independently, just with a smaller perturbation space. In this paper, we take an alternative stance where we produce adversarial examples *while complying with the domain constraints*.

3. Methodology

In this section, we explain the techniques used in adversarial machine learning (AML) for crafting adversarial examples, describe how we adapt them for network intrusion detection systems, and discuss our approaches used in the evaluation section.

3.1. AML for NIDS

Throughout, both in this section for examples and in the evaluation, we focus principally on network intrusion detection data (specifically, the use of TCP/IP). Network data is particularly compelling to use,

[1]Most attack algorithms have upper limits on the allowable distortion they can introduce. This distortion is defined to be the distance between an adversarial example and its original counterpart. There are many metrics (principally l_p norms [5]) used throughout the literature to measure this distance.

as: (1) the underlying protocols encode complex constraints, (2) features are sourced from varying layers in the network stack (adding implicit constraints across layers), and (3) networks serve as a canonical domain for benchmarking practical uses of machine learning in security contexts. Here, we assume a network intrusion detection system classifies feature vectors (representing network traffic flows) as benign or malicious[2]. These feature vectors are created through a feature extraction algorithm, which accepts network packets as input. Therefore, the adversary seeks to bypass the network intrusion detection system by tweaking malicious network traffic (guided by adversarial machine learning techniques), so that it is subsequently classified as benign. Naturally, the adversary must obey the TCP/IP protocol so that the network attacks can be realizable. Any feature vector that violates the TCP/IP protocol (such as negative port numbers) would be manifestly adversarial.

3.2. Challenges in AML in Networks

Crafting adversarial examples for NIDS is a necessarily different process from crafting adversarial examples against image recognition systems. Not all features represent the same kind of information (pixels vs packet information), nor do they describe the same kind of statistical data (discrete vs a blend of categorical, continuous, and discrete features). These differences change the threat surface and the underlying assumptions surrounding the capabilities of an adversary attacking a network intrusion detection system. Existing algorithms are unsuitable for attacking NIDS for these reasons:

(1) Existing algorithms are largely optimized for human perception. While there is an open discussion on the amount and kinds of distortion that are appropriate [5], existing algorithms have been tuned for image domains [5, 12, 29]. That is, these algorithms try to minimize human perception of the distortion introduced in adversarial examples. However, such metrics have no meaning for many domains, including networks, because they are not perceived by humans. Thus, using algorithms optimized for human perception offers us limited utility, especially for network data.

(2) Existing algorithms assume adversaries have full control over the feature space. Most algorithms perturb the entire feature space to minimize an l_p-norm as a surrogate for measuring human perception [5, 12, 29]. This an unreasonable assumption many security-centric domains, especially networks. For example, in network intrusion detection, features can represent broad network behaviors that exist outside the control of an adversary, e.g., round-trip times.

(3) Existing algorithms do not consider domain constraints. Crafting adversarial examples that obey domain constraints is necessary to mount practical attacks. However, prior works strategically applied perturbations in order to *explicitly avoid constraints* [13, 22] (such as only adding bytes at the end of a binary). While a reasonable strategy for malware, there are certainly applications of machine learning where constraint resolution cannot be trivially avoided. As an example of a constraint, network intrusion detection datasets commonly have protocol and service (port number) as features and certain services are exclusive to certain protocols. Therefore, to produce an adversarial example that is representative of a legitimate traffic flow, algorithms need to enforce these constraints.

3.3. Crafting with Network Constraints

In this subsection, we describe how we extract network constraints and incorporate them into crafting adversarial examples.

[2]Without loss of generality, the models we use in the evaluation discriminate between different kinds of network attacks (as well as benign traffic). Nonetheless, the goal of the adversary remains unchanged.

Extracting Network Constraints. If the domain constraints are not explicitly given, then they must be inferred. One of the first requirements for properly modeling the constraints is to codify the specific relationships between features. One of the most important of these relationships is the notion that we introduce as a *primary feature.* Intuitively, a primary feature is a feature that, when set to particular value, limits the range of permissible values for other features. Conversely, a *secondary feature* does not impose any limitations for other features. Therefore, we first identify primary features and their relationships to secondary features.

The concept behind primary features is harmonious with networks in particular. Many popular network intrusion detection datasets include features that represent services, flags, and other protocol-related information. Since these features share a causal relation with protocols, using transport layer protocols to be primary features is intuitive, as we then extract the relationships across many secondary features used in NIDS whose values *directly depend* on the protocol used (such as services and packet flags). Table 5 in the Appendix shows the constraints for one of the network datasets.

Given that constraints are restrictions on where and how features can be perturbed, it is intuitive to model these constraints as a simple form of first-order logic. Here, primary features are the predicates in logical expressions that determine the properties (e.g., values) that a collection of variables (e.g., secondary features) can have. Said alternatively, the set of primary features are the conditionals on the values of other secondary features. Formally, we can understand constraints to have the following form:

$$\forall \boldsymbol{x} \in \mathbb{X} : \boldsymbol{x}_k \Rightarrow (\boldsymbol{x}_1 \in \mathbb{Y}_1) \wedge (\boldsymbol{x}_2 \in \mathbb{Y}_2) \wedge \cdots \wedge (\boldsymbol{x}_n \in \mathbb{Y}_n)$$

where $\boldsymbol{x}$ represents an input in a dataset $\mathbb{X}$, k is a primary feature, and $\mathbb{Y}_n$ represents the values permissible by the semantics of feature n (e.g., $\{0,1\} \in \mathbb{Y}$ if the feature is binary, or $\{\mathbb{R} : 0 \leqslant y \leqslant 1\} \in \mathbb{Y}$ if the feature is continuous). For example, we can represent a network constraint between TCP and ports as follows:

$$\forall \boldsymbol{x} \in \mathbb{X} : \boldsymbol{x}_{TCP} \Rightarrow \boldsymbol{x}_{port} \in [1, \ldots, 65535]$$

After primary features have been identified, we begin to extract constraints based on the following heuristic: a constraint exists between a primary feature k, and any other feature p, if there exists at least one input in the training set where both k and p are seen together. For example, features that describe TCP packet flags (i.e., p) would have the value 1.0 for TCP traffic flows (i.e., k) and 0.0 for non-TCP traffic flows. Therefore, TCP packet flag features are *constrained* to TCP flows. Conceptually, these constraints encode the maneuvers that are possible (and probable) for an adversary.

Integrating Network Constraints. To address the challenges discussed prior, we integrate *constraint resolution* into the crafting process. This guarantees that generating an adversarial example obeys not only the semantic constraints of the network domain, but also the probabilistic constraints of the dataset as well. As an artifact of our constraint learning process, we may also learn constraints that are technically allowable in networks, but are simply never observed in the dataset. As a second technique, we simultaneously minimize the total number of features perturbed to obey constraints and comply with the limited control over features an adversary may have.

Measuring Distance. The distance between an adversarial example and its original counterpart is a measurement of the distortion introduced by an algorithm, commonly represented by an l_p norm. Most algorithms have either limits on the maximum allowable distortion or explicit termination conditions when a particular amount of distortion is introduced. There is debate on the most appropriate distance measure (i.e., the choice of l_p norm) to use for modeling levels of human perception [5]. Our study in the network domain departs from this debate, as it is not inherently visual. We argue that, for non-visual domains, greater insight can be gained on adversarial effectiveness when parameterizing for *the number of features* under control by the adversary, versus parameterizing for the maximum amount that they can perturb *every* feature. For our evaluation on the efficacy of adversaries against NIDS, measuring distance under the l_0 norm is the most appropriate choice among the other norms used in attacking image recognition systems.

Augmenting the JSMA. Before we can use the JSMA for network data, we improve the algorithm so that it can better search the space of possible adversarial examples. By default, the JSMA parameter θ determines the magnitude and direction of a selected perturbation. A positive θ will increase features and a negative θ will decrease them. To allow an adversary more freedom when crafting adversarial examples, we modified the JSMA to perturb in either direction dynamically, which directly improved the success rate of the JSMA. We refer to this modified version as the A(DAPTIVE) JSMA.

To allow the AJSMA to perturb in either direction, we evaluate both masks used in [36] that are applied to the saliency map when θ is positive or negative. Formally, for any feature i to be a perturbation candidate, i must satisfy:

$$\left(\frac{\partial f_t(\boldsymbol{x})}{\partial \boldsymbol{x}_i} > 0 \text{ and } \sum_{j\neq t} \frac{\partial f_j(\boldsymbol{x})}{\partial \boldsymbol{x}_i} < 0\right) \text{ or } \left(\frac{\partial f_t(\boldsymbol{x})}{\partial \boldsymbol{x}_i} < 0 \text{ and } \sum_{j\neq t} \frac{\partial f_t(\boldsymbol{x})}{\partial \boldsymbol{x}_i} > 0\right)$$

where $\frac{\partial f_t(x)}{\partial \boldsymbol{x}_i}$ represents the forward derivative for a model f and target class t with respect to feature i in an input $\boldsymbol{x}$. Conceptually, the AJSMA only considers features to be perturbable if the target gradient and sum of non-target gradients are in opposing directions. Intuitively, this means that any perturbation reduces the distance to the target class or increases the distance to non-target classes.

This modification enables us to determine the optimal perturbation direction for increasing (the left half of the mask) or decreasing (the right half of the mask) features. Afterwards, we use the scoring metric found in [36] and return the most influential feature and optimal perturbation direction. Finally, the algorithm terminates when the input is misclassified or the l_0 norm budget is exhausted.

Integrating Network Constraints into the AJSMA. Integrating constraint resolution distills to preventing the AJSMA from selecting features that violate constraints. Once a feature is selected for perturbation, we check if this feature is constrained to a primary feature. This check is described by Algorithm 1.

Consider the following example using the Algorithm 1. A UDP traffic flow is given to the AJSMA. After analyzing the saliency map, the AJSMA suggests that the current service, TFTP_U, should be switched to FTP, which is a service constrained to TCP. After the service switch is made, the perturbed input is presented to Algorithm 1 to check whether or not any constraint is violated. The first condition determines if the perturbed feature p is a primary feature (i.e., TCP, UDP, or ICMP). In this example, it is not, so we move to the second condition and evaluate if p is constrained to exclusively one primary feature. Here, p is exclusively associated with TCP. Thus, the search domain is further restricted to TCP-compliant features and the transport protocol of the input is switched from UDP to TCP. Since the input has switched primary features from UDP to TCP, we finally set all non-TCP features to 0. Once the AJSMA terminates, the produced adversarial example will be representative of a permissible traffic flow, i.e., it obeys the domain constraints.

Algorithm 1: RESOLVING CONSTRAINTS

p is a candidate feature, Γ is the search domain, $\boldsymbol{S}$ is the saliency map for the current input $\boldsymbol{x}$, $h : \mathbb{K} \mapsto \mathbb{V}$ is an associative array containing constraints.

```
Input: $p$, $\Gamma$, $\boldsymbol{S}$, $\boldsymbol{x}$, $h$
// p is a primary feature
if $p \in \mathbb{K}$ then
    $\Gamma = \Gamma \cap h(p)$
    $\boldsymbol{x}_p \leftarrow$ switch primary feature to $p$
end
// p is constrained to exactly one primary feature
else if $\exists! k \in \mathbb{K}$ s.t. $p \in h(k)$ then
    $\Gamma = \Gamma \cap h(k) \setminus \{p\}$
    $\boldsymbol{x}_k \leftarrow$ switch primary feature to $k$
end
// p is constrained to multiple primary features
else if $\exists k \in \mathbb{K}$ where $p \in h_k$ then
    $\Gamma = \Gamma \setminus \{p\}$
    $k' \leftarrow$ the current primary feature in $\boldsymbol{x}$
    // x is using an illegal primary feature wrt p
    if $p \notin h(k')$ then
        $k = \arg\max_{\{k \in \mathbb{K} | p \in h(k)\}} S_k$
        $\Gamma = \Gamma \cap h(k)$
        $\boldsymbol{x}_k \leftarrow$ switch primary feature to $k$
    end
end
// p constrains all primary features
else if $\forall k \in \mathbb{K}, p \in h_k$ then
    $\Gamma = \Gamma \setminus \{p\}$
end
if switched primary feature to $k$ then
    // ensure x is not using illegal features
    $\forall i \notin h(k), \boldsymbol{x}_i \leftarrow 0$
end
return $\Gamma$, $\boldsymbol{x}$
```

3.4. *Creating Adversarial Sketches*

In this subsection, we describe how we create adversarial sketches[3]: universal perturbations that obey domain constraints.

For the same reasons described at the beginning of §3, existing algorithms to generate universal adversarial perturbations [4, 16, 30] are not immediately usable in networks: they are optimized for human perception, assume adversaries have full control over the feature space, and do not consider domain constraints.

In our search for adversarial sketches in networks, we take a principled approach using two tools: adversarial examples generated from the AJSMA and a *perturbation histogram*. Prior to the algorithms mentioned earlier for computing universal adversarial perturbations, encountering these perturbations was a matter of chance: an adversary would be required to apply a perturbation generated from an attack on other unperturbed inputs and observe the universality of the perturbation (i.e., brute-force) [12].

Adversarial Examples from the AJSMA. Initially, we followed the same brute-force approach in [12]: we used adversarial examples crafted from the AJSMA to glean insights for universal perturbations. This brute-force approach is feasible (in terms of computational complexity) for network intrusion detection datasets as their dimensionality and cardinality is small. After evaluating every perturbation generated by the AJSMA on every input in the test set, we discovered a handful of universal perturbations.

[3]The concept of "sketching," also known as *approximate query processing* [7], was first introduced by Flajolet et al. Sketching refers to a class of streaming algorithms that seek to extract information from a data stream in a single pass [10]. Commonly deployed in memory-constrained environments, these algorithms approximate or summarize the information in a given data stream. Adversarial sketches are similar as they are an approximation of a universal perturbation and computed through one pass of inputs.

Perturbation Histograms. The perturbation histogram encodes how perturbations are distributed en masse (an example is shown later in §4). To produce the histogram, we enumerate over all of the perturbations generated by the AJSMA (including any additional perturbations made to resolve constraints) and record the perturbed features and directions. Our insight for the histogram is rooted in how the AJSMA scoring metric ranks influential features; we hypothesized that features commonly perturbed across inputs from different classes would be optimal candidates for building an adversarial sketch. This hypothesis was reinforced by our observation that the perturbation histogram was (relatively) static: random shuffling of partitioned training sets, unique training parameters, and random subsets of analyzed adversarial examples yielded minor changes to the perturbation histogram. These substantial adjustments to our experiment workflow demonstrated little change between perturbation histograms. These observations suggest that the perturbation histogram is a combined representation of class-based saliency *and* domain constraints.

With the universal perturbations discovered through the AJSMA and the static nature of the perturbation histogram, we made an observation: **the majority of the perturbed features in the most successful universal perturbations generated from the AJSMA mapped *directly* to the most perturbed features in the perturbation histogram. This key observation led us to the creation of the *Histogram Sketch Generation*.**

Histogram Sketch Generation. The HISTOGRAM SKETCH GENERATION accepts a perturbation histogram $\boldsymbol{H}$ and integer n as parameters and returns a adversarial sketch $\boldsymbol{a}$, which consists of the top n most frequently perturbed features and optimal directions[4]. We observed that the most successful universal perturbations generated by the AJSMA had a subset of features that directly mapped to the most frequently perturbed features in the perturbation histogram. Intuitively, if we consider these successful universal perturbations as the optimal solution, then the HSG approximates the optimal solution by returning a subset of those features (and associated directions). Figure 6 in the Appendix demonstrates a handful of sketches.

It is interesting to note that the HISTOGRAM SKETCH GENERATION is essentially the problem of variable selection in classical statistics. Variable selection involves the selection of a subset of relevant variables (or features) in the model, such that a "minimal" amount of information is lost (e.g., minimal impact on model loss). Many common procedures in classical statistics, such as LASSO or stepwise regression, aim to balance model fit with a penalty on the l_0 norm (or a relaxation of this norm to l_1, in the case of LASSO)[48]. Our approach of greedily selecting the top n most perturbed features is directly analogous to stepwise regression's (greedy) selection of the n features which best explain the data. As the complexity grows combinatorially with the number of features, greedy methods are necessarily resorted to, and our approach here is no exception.

4. Evaluation

In this section, we evaluate our approach on two network intrusion datasets, the NSL-KDD and the UNSW-NB15, and two image recognition datasets, the GTSRB and MNIST. We answer two questions:

(1) Do constraints add robustness against adversarial examples?
(2) Do universal adversarial perturbations exist?

[4]While the HSG does not take a target class as a parameter, it uses information directly from the perturbation histogram to create an adversarial sketch. As a consequence, the HSG will return a targeted adversarial sketch if the histogram is built from targeted adversarial examples.

The experiments showed that we can craft adversarial examples with success rates greater than 95%, even with domain constraints; we can compute successful adversarial sketches, reaching greater than 80% misclassification rates for many learning techniques.

Our experiments were performed on a Dell Precision T7600 with an Intel Xeon E5-2630 and a NVIDIA GeForce TITAN X. We used CleverHans [34] for crafting adversarial examples.

4.1. Datasets

Before we describe our experiments, we provide an overview of the four evaluated datasets and any preprocessing that we performed. Table 1 describes the model architectures, hyperparameters, and model accuracies across all four datasets.

To measure the impact of network constraints on adversaries, we evaluate our approach on two network intrusion detection datasets and compare our results against two image classification datasets. While our image datasets do not contain constraints, we use them to: (1) compare the cross-domain generalization of our attacks (that is, the AJSMA and the HSG), and (2) provide a visualization of our techniques, which we detail in the Appendix 7

NSL-KDD. The NSL-KDD dataset is an improved variant of the KDD Cup99 dataset [47]. The KDD Cup99 (and its NSL-KDD successor) have been used widely in the network intrusion detection community. We chose to use the NSL-KDD for the novel application of adversarial examples in this field and familiarity of the dataset within the academic community.

The NSL-KDD contains 5 classes, with 4 attack classes and 1 benign class. It contains 125,973 samples for training and 22,543 samples for testing. It contains 41 features[5], separated into four high-level categories of features: basic features of TCP connections, content features within a connection suggested by domain knowledge, traffic features that are computed using a two-second time window, and host-based features. The NSL-KDD has been widely studied and so we defer to prior work [8, 47] for the subtle details of the dataset. The state-of-the-art accuracy reported is 77.41% with Multi-layer Perceptrons [1, 18, 19, 33, 47].

UNSW-NB15. The UNSW-NB15 dataset was designed to be an updated version of the NSL-KDD, containing modern attacks that express a "low footprint" [31]. The Australian Centre for Cyber Security (ACCS) used the *IXIA PerfectStorm* tool to create a combination of normal and abnormal network traffic. The abnormal traffic generated from the IXIA PerfectStorm tool is broken down into nine attack types, which we used as our source classes for generating our adversarial examples. After the traffic is generated, the authors leveraged *Argus* and *Bro-IDS* tools to construct reliable features.

The UNSW-NB15 contains 10 classes, with 9 attack classes and 1 benign class. It contains 175,341 samples for training and 83,332 samples for testing. The dataset contains 48 features, separated into four

Table 1
Model Information

Dataset	Architecture	Units	Batch Size	Learning Rate	Epochs	Testing Acc.
NSL-KDD	MLP	123, 64, 32, 5	200	0.01	5	77% $\pm$ 1.0%
UNSW-NB15	MLP	196, 98, 49, 10	128	0.01	10	75% $\pm$ 1.2%
MNIST	CNN	784, 128, 128, 10	128	0.001	6	98% $\pm$ 0.1%
GTSRB	CNN	2700, 128, 128, 42	128	0.001	6	82% $\pm$ 2.0%

high-level categories of features: flow-based, basic connection, content, and time-based. We defer to the authors for a comprehensive description of the dataset [31] and its similarity with the NSL-KDD [32, 33]. The state-of-the-art accuracy reported is 81.34% with an artificial neural network [28, 32].

MNIST. The Modified National Institute of Standards and Technology database contains hanwritten digits. We chose to use MNIST due to its popularity in the adversarial machine learning community and to serve as a comparison to the network data as an unconstrained application of adversarial examples.

The MNIST database contains 10 classes, with numerical digits from 0 through 9. It contains 60,000 samples for training and 10,000 samples for testing. No preprocessing was required to integrate this dataset into our experimental setup. We defer to the authors for the intricate details surrounding the MNIST dataset [24]. The state-of-the-art accuracy reported is $\sim$99% [39].

GTSRB. The German Traffic Sign Recognition Benchmark is a dataset of common traffic signs found throughout Germany. We chose to use the GTSRB as a second unconstrained dataset to compare against the network intrusion detection datasets.

The GTSRB contains 42 classes. After preprocessing, our experiments contained 21,792 samples for training and 6,893 samples for testing. Throughout the dataset, there are identical images of varying sizes. We first cropped the region of interest (which contains the traffic sign) and downsampled to a final size of 30×30. For more details concerning the GTSRB, we defer to the authors [44]. With data augmentation, input preprocessing, and special network architectures, the state-of-the-art accuracy reported is over 97% [25].

4.2. Experiment Overview

In this subsection, we describe our experiments in detail, using the NSL-KDD as an application of our approach.

Through our evaluation on network intrusion detection, we note that our experiments emulate a realistic adversary by taking different attacks and masking them as benign traffic.

Data Curation. We perform a stratified shuffle-split on the training set into five parts. This split lays the foundation for measuring transferability: each partition (which we refer to as *A, B, C, D, E*) is representative of a uniquely trained model (which we refer to as M_A, M_B, M_C, M_D, M_E respectively), mirroring the setup described in [37]. This setup allows us to measure *intra-technique* transferability rates: the rate at which adversarial examples crafted from one model are also misclassified by another model with the same learning technique. Also, we can also measure the converse, *inter-technique* transferability: the same misclassification rate except with a model of an different learning technique[6].

Next, we perform a stratified shuffle-split on the test set. This creates two sets of isolated inputs: we use the AJSMA on one set and the HSG on the other. This is to ensure that the sketches are only applied on inputs that had no influence in the creation of the sketch (recall that we analyze the adversarial examples created by the AJSMA to build sketches). Furthermore, the stratified shuffle-split of the test set is to ensure that the AJSMA crafts adversarial examples from inputs spanning all classes. This enables us to create effective adversarial sketches, as all class-specific information is distilled into the perturbation histogram.

Using the NSL-KDD as an example for the first stage, we built a Multi-layer Perceptron with 4 layers: an input layer of 123 units, fully-connected to 64 units, fully-connected to 32 units, and finally an output

[6]For our inter-technique evaluation, we consider four popular learning techniques: Logistic Regression (LR), Decision Trees (DT), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN). Each one of these learners represent different learning paradigms (and are popular in commercial and academic contexts), and are thus appropriate candidates for evaluating inter-technique transferability. Hyperparameters and other details can be found in the Appendix.

Table 2
NSL-KDD constraint distribution, categorized by feature type - Unlike TCP, UDP and ICMP have limited degrees of freedom.

	Feature Type				
Protocol	Basic	Content	Timing-based	Host-based	Total
TCP	81	12	9	10	112
UDP	12	0	7	8	27
ICMP	14	0	7	8	29

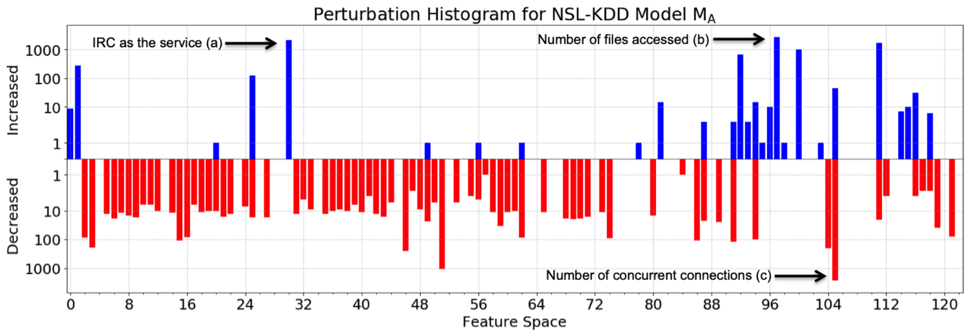


Fig. 1. NSL-KDD model M_B Perturbation Histogram produced by adversarial examples from the AJSMA in log scale - Certain features are *consistently* increased (a) & (b) and decreased (c), indifferent of the source class.

layer with 5 units[7]. The output layer conveys our 5 classes: Normal (Benign), Probe, Denial of Service (DoS), User to Root (U2R), and Remote to Local (R2L). We used rectified linear units (ReLU) as our chosen activation function for our hidden layers and softmax at the output layer. Our models are trained via the Adam optimizer [21] with a batch size of 200 and a learning rate of 0.01 for 5 epochs. With our five splits, each model, M_A through M_E, is trained with ∼25,194 inputs. With these hyperparameters and network architecture, we were able to achieve an average 77% ± 1.0% accuracy on the test set, which is consistent with the literature [18, 19, 33, 47].

Constraint Extraction. With the heuristic described in §3, we extract constraints in the NSL-KDD with PROTOCOL as the primary feature. The intuition behind this selection is straightforward: a majority of the features in the NSL-KDD describe metadata surrounding these protocols, e.g., flag information, services, and content-related features like FTP commands. The distribution of the extracted constraints for the NSL-KDD can be found in Table 2. We observe that the TCP protocol offers the highest degree of maneuverability by a wide margin (and to no surprise as it constitutes the majority of traffic flows in the dataset). This is unlike UDP and ICMP, who are significantly more constrained. Table 5 in the Appendix shows all of the constraints, sorted by feature type.

Adversarial Example Generation. To investigate the first question in our evaluation, we craft adversarial examples with the AJSMA. With the network constraints integrated into the crafting process, we iterate over the first half of the test set for each model, M_A through M_E, and craft adversarial examples (the

[7]We note that the number of layers and units was influenced by research that suggests an optimal upper bound for the number hidden neurons for feed-forward networks [17]. The remainder of our hyperparameter selection follows no formal process.

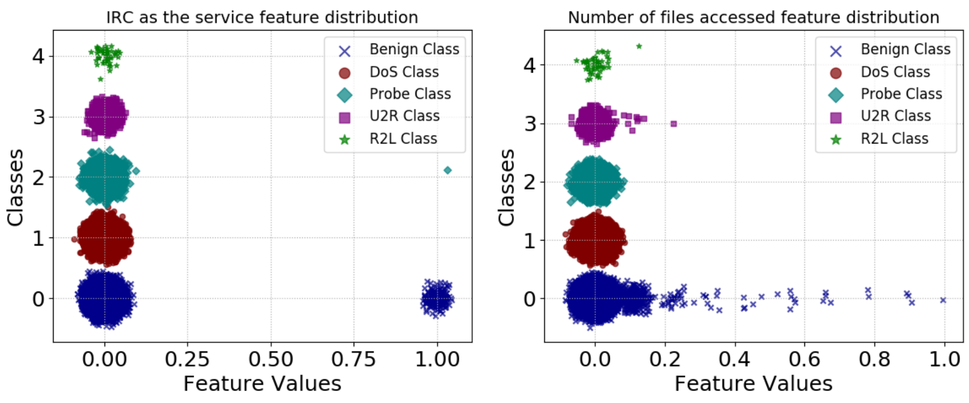


Fig. 2. NSL-KDD class distribution for SERVICE=IRC (left) and NUM_ACCESS_FILES (right) - These two features are often perturbed due to their bias towards the "Benign" class, shown as class 0 on the Y-axis.

second half of the test set is used by the HSG to craft adversarial examples through sketches). Table 3 describes the output of this stage from an example NSL-KDD run with "Benign" as the target class. Figure 1 shows the perturbation histogram computed from the same run.

We note that there are particular features that were consistently perturbed in nearly all adversarial examples, namely setting the service as IRC and increasing NUM_ACCESS_FILES[8]. To understand why, we used WEKA [15] to analyze the distribution of these features. The analysis showed that inputs which use IRC as the service and have high values for NUM_ACCESS_FILES are heavily skewed towards the target class, "Benign". Figure 2 shows these distributions (the "Benign" target class is bottom class (0) on the Y-axis.).

Adversarial Sketch Generation. To investigate the second question in our evaluation, we use the perturbation histogram (computed from adversarial examples crafted by the AJSMA) to create adversarial sketches. As described in §3, the HSG creates adversarial sketches by selecting the top n most perturbed features from the perturbation histogram. With the top n most perturbed features, we craft adversarial examples by applying the sketch to the second half of the test set, for each model, M_A through M_E.

[8]While it may seem this feature is not derived from network data, both the NSL-KDD and UNSW-NB15 have a "high-level content" category of features, which are derrived from payload information.

Table 3

Output from crafting adversarial examples with the AJSMA for NSL-KDD model, M_B - Even domains with constraints are vulnerable to adversarial examples.

AJSMA Experiment Results - Target 0 "Benign"	
Testing Inputs	11,272
Labeled as Target Class	4,856
Misclassified as Target Class	2,532
Number of Inputs Attacked	3,884
Average Distortion	3.39% ∼4.18 features
Class Success Rates	DoS:100%, Probe:99%
	U2R:3.99%, R2L:100%

4.3. Measuring Success

With the adversarial examples crafted via the AJSMA and HSG, we measure the effectiveness of the two algorithms through white-box attacks and through transferability. We define the success rate of white-box attacks as the number of adversarial examples misclassified as the target class over the number of attempted inputs:

$$SR_{wb} = \frac{|\{\boldsymbol{x} \in \mathbb{X} : f(\boldsymbol{x}) = t\}|}{|\mathbb{X}|}$$

where $\mathbb{X}$ represents the set of attempted inputs, f is a model, and t is the target class. Furthermore, we define the transferability success rate to be the number of adversarial examples misclassified as the target class by the target model over the number of successful adversarial examples crafted from the source model, formally:

$$SR_{transfer} = \frac{|\{\boldsymbol{x} \in \mathbb{X} : f'(\boldsymbol{x}) = t\}|}{|\{\boldsymbol{x} \in \mathbb{X} : f(\boldsymbol{x}) = t\}|}$$

where $\mathbb{X}$ again represents the set of attempted inputs from the source model f, and f' represents the target model. These measurements of success are used ubiquitously in both white-box and grey-box threat models [6, 13, 22, 35] and our results corroborate those with similar L_0 distortion budgets [37].

We argue that measuring the accuracy of ML-based NIDS is appropriate for security validation, as accuracy gives network operators a metric by which to measure how the predictions of the NIDS *conform* to the expectation of network operators. Specifically, since ML-based NIDS rely on a labeled dataset from which to train, these labels are either (1) provided directly by network operators, or (2) provided through traffic analysis tools, such as IXIA Perfect Storm [31]. Thus, the efficacy of ML-based NIDS to produce predictions that conform to the provided labels aligns with expectations of security evaluations of such systems. Moreover, it also serves as the standard measurement in adversarial machine learning evaluations [5, 11, 30, 36] and guides us to understand how adversaries would lay out attacks practically.

Specifically, an adversary would lay out an attack as follows: (1) the adversary would first use the AJSMA on a set of inputs to craft adversarial examples from, (2) with this set of adversarial examples, a perturbation histogram can be computed to determine the most influential set of features for the model under attack, (3) the adversary would then select the top-n[9] features, as determined by the perturbation histogram, to produce an adversarial sketch, and (4) with an adversarial sketch, the adversary would apply the sketch to a set of desired inputs and attack the model. We report the efficacy of the AJSMA and HSG in Table 4.

AJSMA Results. In Table 4 (a) on the left, we show the NSL-KDD AJSMA success rates for white-box attacks and transferability rates. The labels on the left represent the source model used to generate adversarial examples and the labels on top represent the target models that were attacked. For the intra-technique case, the white-box results can be read along the diagonal (as the source and target are the same model). For both intra- and inter-technique cases, the transferability rates are represented in all other cells. The AJSMA was broadly successful in creating targeted adversarial examples while introducing relative amounts of distortion comparable to image-based experiments, even in the presence of constraints.

[9]The choice of n is a function of the adversarial threat model; for our evaluation, we select an n for each dataset that roughly represents ≈ 4 of l_0 distortion. This threat model is similar with those studied in the literature, as discussed in Section 2.

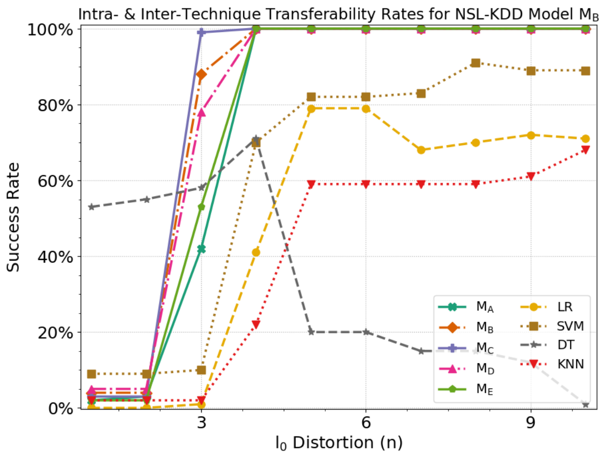


Fig. 3. HSG intra- and inter- transferability rates for NSL-KDD model M_B for target "Benign" as a function of l_0 distortion n - Values between 6-9 for n have greater than 70% misclassification for most learning techniques.

These results suggest that the constraints for our evaluated domain do not offer any robustness against adversarial examples.

Additionally, it is interesting that the adversarial examples produced by the AJSMA had notable transferability rates in intra-technique case (an average of 73% across our network intrusion detection experiments). This would suggest that transferability is stronger among ML-based network intrusion detection systems.

HSG Results. In Figure 3, we show NSL-KDD model M_B success rates for intra- and inter-technique transferability for varying values of n between 1 and 11. There are broad regions for values of n which have high success rates, reaching 100% in white-box settings and greater than 80% transferability rates for the majority of learning techniques. Indeed, this suggests that the most often perturbed features (by the AJSMA) are appropriate candidates for building an adversarial sketch. Furthermore, these results confirm the existence of universal adversarial perturbations that obey domain constraints.

Finally, we are surprised at the fragility of the models trained on network intrusion detection data; sometimes perturbing only *three* features were necessary to misclassify greater than 80% of inputs in the test set. We believe that this fragility is partly a function of the skewed distribution certain features can have for specific classes, such as the ones shown in Figure 2. This insight is unlike the adversarial examples crafted against image recognition systems, where high dimensionality and a more balanced class distribution for features appear to mitigate this fragility.

Table 4 shows the success rates for both of our algorithms across all four datasets. The values of n listed equal $\sim 4\%$ l_0 distortion for the HSG. Again, the labels on the left represent the source model used to generate adversarial examples, while the labels on top represent the target models that were attacked. For the intra-technique case, the white-box results can be read along the diagonal (as the source and target models are the same). The transferability rates, for both intra- and inter-technique cases, are represented in all other cells.

We observe high success rates for both the AJSMA and the HSG in both white-box attacks and transferability for several of the datasets. While the AJSMA struggled significantly to produce adversarial examples that transferred as the dimensionality increased, the HSG transferability rates were more resilient (but still affected) to the increased dimensionality[10]. This would suggest that the distance between decision boundaries increases as model complexity increases, thus mitigating transferability for our algorithms.

Moreover, we note that the inter-technique transferability rates were largely model-dependent. Specifically, we observe that relatively low transferability rates using the AJSMA, while the HSG maintains high transferability rates for the network datasets. This is likely due to the inherit design of the AJSMA—since the attack terminates *immediately* after an adversarial example is misclassified as benign traffic, we posit that adversarial examples are not sufficiently over the decision manifold for a source model that would also cross the decision manifold for other target models (since their decision manifolds would be slightly different). These results corroborate inter-technique transferability of adversarial examples produced by the JSMA in other experiments [37]. While we emphasize our evaluation principally on white-box attacks, we hypothesize that producing additional iterations of perturbations would raise the transferabiltiy rate, albeit at the likelihood of increased distortion, due to enforcing domain constraints.

ADAPTIVE JSMA

	M_A	M_B	M_C	M_D	M_E	LR	SVM	DT	KNN
M_A	100%	69%	51%	51%	61%	34%	40%	34%	20%
M_B	73%	99%	72%	78%	67%	49%	50%	39%	44%
M_C	63%	69%	100%	66%	69%	30%	34%	27%	24%
M_D	54%	93%	70%	99%	64%	24%	38%	25%	22%
M_E	83%	79%	71%	66%	100%	42%	55%	41%	17%

HISTOGRAM SKETCH GENERATION

$n = 6$	M_A	M_B	M_C	M_D	M_E	LR	SVM	DT	KNN
M_A	100%	100%	100%	100%	100%	80%	85%	16%	19%
M_B	100%	100%	100%	100%	100%	79%	82%	20%	59%
M_C	100%	100%	100%	100%	100%	65%	83%	50%	34%
M_D	100%	100%	100%	100%	100%	81%	87%	22%	59%
M_E	100%	100%	100%	100%	100%	98%	93%	27%	19%

(a) NSL-KDD

	M_A	M_B	M_C	M_D	M_E	LR	SVM	DT	KNN
M_A	100%	97%	92%	96%	96%	72%	81%	29%	53%
M_B	50%	100%	72%	94%	71%	62%	62%	12%	26%
M_C	73%	81%	100%	93%	87%	71%	76%	19%	49%
M_D	69%	64%	55%	100%	59%	53%	48%	8%	25%
M_E	66%	80%	90%	96%	100%	66%	69%	15%	38%

$n = 9$	M_A	M_B	M_C	M_D	M_E	LR	SVM	DT	KNN
M_A	88%	99%	95%	96%	96%	99%	96%	29%	37%
M_B	99%	100%	99%	100%	99%	99%	100%	20%	62%
M_C	93%	97%	100%	77%	95%	73%	100%	27%	41%
M_D	80%	99%	100%	100%	99%	74%	99%	13%	30%
M_E	98%	100%	100%	94%	92%	85%	100%	28%	39%

(b) UNSW-NB15

	M_A	M_B	M_C	M_D	M_E	LR	SVM	DT	KNN
M_A	100%	39%	30%	19%	11%	21%	24%	18%	5%
M_B	6%	100%	14%	6%	2%	19%	18%	14%	2%
M_C	12%	32%	99%	9%	8%	23%	22%	15%	1%
M_D	17%	44%	26%	100%	21%	21%	21%	19%	5%
M_E	20%	53%	33%	20%	99%	24%	27%	18%	6%

$n = 41$	M_A	M_B	M_C	M_D	M_E	LR	SVM	DT	KNN
M_A	24%	35%	18%	14%	9%	45%	48%	14%	7%
M_B	17%	42%	14%	15%	54%	62%	61%	13%	5%
M_C	22%	31%	41%	9%	14%	60%	55%	32%	5%
M_D	13%	30%	14%	36%	13%	46%	46%	33%	6%
M_E	15%	29%	11%	12%	31%	44%	58%	21%	4%

(c) MNIST

	M_A	M_B	M_C	M_D	M_E	LR	SVM	DT	KNN
M_A	97%	2%	1%	3%	2%	0.2%	1%	3%	0%
M_B	1%	95%	0.5%	1%	1%	0.3%	2%	2%	0%
M_C	2%	2%	98%	2%	0.8%	0.1%	2%	1%	0%
M_D	0.5%	1%	3%	99%	1%	0.2%	0.8%	3%	0%
M_E	0.7%	2%	1%	2%	94%	0.1%	0.5%	2%	0%

$n = 104$	M_A	M_B	M_C	M_D	M_E	LR	SVM	DT	KNN
M_A	34%	6%	27%	6%	7%	0.2%	2%	12%	0%
M_B	9%	27%	14%	2%	8%	0.5%	1%	6%	0%
M_C	17%	4%	44%	2%	7%	0.6%	1%	5%	0%
M_D	29%	12%	16%	29%	12%	1%	1%	5%	0%
M_E	14%	8%	15%	3%	35%	0.1%	1%	4%	0%

(d) GTSRB

Table 4

Results for AJSMA (left) and HSG (right) for all of our experiments for target class “Benign” - Values of n for our sketches represent $\sim 4\%$ l_0 distortion.

[10]We hypothesize that this is partly due to how the AJSMA is designed. Conceptually, the AJSMA creates adversarial examples that *just* cross over the decision boundaries. While the decision boundaries among models of lower dimensionality may be similar (and thus, an adversarial example can cross over the decision boundaries of multiple models), this appears to be untrue for models of higher dimensionality, where there can be multiple unique ways to separate the data.

From our investigation, we highlight some key takeaways:

(1) Network constraints are not problematic for crafting adversarial examples in the datasets we studied. The AJSMA reached 100% success rates for most attempted inputs, with distortion rates comparable to image-based experiments, while in the presence of constraints.
(2) Universal adversarial perturbations that obey network constraints exist. The HSG produced Adversarial Sketches that reached 100% success rates in network intrusion detection applications for a values of n that represent $\sim 4\%$ l_0 distortion. Furthermore, they produced adversarial examples that had higher transferability rates than the AJSMA.
(3) Network intrusion detection data is highly fragile: a small dimensionality and biased distributions enable attack algorithms to alter very few features to successfully craft targeted adversarial examples that obey constraints.
(4) Worst-case scenarios (i.e., white-box attacks) are highly vulnerable, not surprisingly. With access to model parameters, adversaries have sophisticated levels of control.
(5) Network intrusion detection data appears to be highly vulnerable to transferability attacks. Even in the presence of disjoint training sets and different learning techniques, both attacks produced adversarial examples with surprising levels (an average of 73% for the AJSMA and 97% for the HSG, for models of the same learning technique) of transferability.

5. Uncontrollable Features

Throughout this paper, we have discussed some of the differences between adversarial machine learning in constrained networks versus unconstrained images. One of the most fundamental questions within this community is: *what precisely is an "adversarial example?"* There are varying definitions with different objectives used throughout AML research. In the image space, it has been generally agreed that if an attack algorithm produces perturbations that are undetectable by a human observer, then it is an adversarial example. However, it is not clear how this translates to other domains.

Research outside of image space (usually) provides their own definitions: perturbed malware must maintain its properties of malware [13, 22], perturbed audio must be nearly inaudible [6], perturbed text must preserve its semantics [9, 20], among other definitions. For our work in network intrusion detection, we follow an intuitive definition: perturbed network flows must maintain their attack behavior. For example, a denial-of-service attack must still deny service to an application or system, post-perturbation.

However, validating attack behaviors is a nontrivial task as security is contextual: DoS attacks on government networks have vastly different behaviors than attacks on small businesses. Therefore, any sort of simulation to observe attack behavior must be in a similar context to the one in which the dataset was built from. This is particularly challenging for old datasets like the NSL-KDD, as even a similar network setup would have different behavior given the modern hardware and software on the systems that would constitute that network. Even if all these factors could be accounted for, there are certain classes of attacks whose "success" is challenging to measure. For example, in reconnaissance attacks it is difficult to know if the information gathered by an adversary is useful.

Regardless, all of these points of contention are driven by a single hypothesis: *Perhaps adversarial examples cannot be crafted if features which represent the semantics of the attack cannot be perturbed.* Instead of attempting to justify why any set of features is critical to the semantics of the attack, we take a different stance on addressing this hypothesis: even if some reasonably sized subset of features could not be perturbed (as to not invalidate the attack semantics), we argue that adversarial examples can still

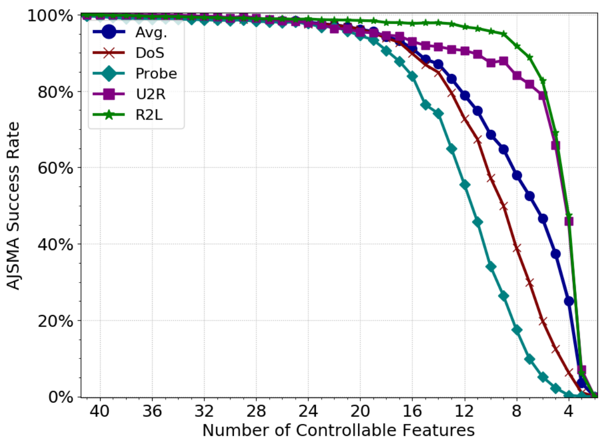


Fig. 4. AJSMA Success Rate with unperturbable features - The overall success rate starts to decrease significantly when the adversary is restricted to controlling $\sim$12 features.

be successfully crafted. With the lessons learned throughout our experiments, we set out to investigate this hypothesis with a straightforward experiment.

To test this hypothesis, we iterate over multiple sets of randomly selected features and make them unperturbable by the adversary. We repeat this random selection for sets of varying cardinalities from 1 to 41 features (where 41 represents the entire feature space). We train a new model on the full NSL-KDD training set and use the 100 most representative[11] inputs from each class[12] from the test set. Next, we iterate over all of the possible combinations of features made unperturbable to an adversary, i.e., $\binom{41}{k}$[13] for $k \in \{1, 2, \ldots, 41\}$[14]. Once we have identified a set of unperturbable features, we eliminate that set from initial search domain of the AJSMA. In this experiment, we crafted a total of 17,664,191 adversarial examples. Figure 4 demonstrates the success rate of the AJSMA as a function of the number of controllable (i.e., perturbable) features.

The results support our argument: the success rate of AJSMA begins to decline when the adversary can only control $\sim$12 features. Even when an adversary has control over only *five random* features (which represents around $\sim$10% of the feature space), the success rate of crafting adversarial examples (with the most representative forms of an attack) is slightly less than 50%. Furthermore, this result demonstrates that applying more restrictive constraints would have little impact on the success of the adversary.Finally, these results suggest that *even if most features which had skewed distributions were removed, adversaries could still craft adversarial examples*. In the Evaluation Section 4, we observed that models were fragile in part due to skewed feature distributions; this experiment demonstrated that this fragility is not isolated to a small handful of features. Thus, approaches such as dimensionality reduction may be ineffective against adversarial examples.

[11] We find the most representative inputs by maximizing the difference (via the softmax layer) between the output component that corresponds to the label and the sum of the components that represent all non-label classes.

[12] The "R2L" class only had 17 inputs that were correctly classified. Thus, we crafted from 317 inputs as opposed to 400.

[13] The NSL-KDD has 41 features before expanding categorical features to one-hot vectors. If a particular combination contains a categorical feature, we eliminate all possible values associated with the feature from the search domain.

[14] To prevent combinatoric explosion, we randomly sampled 1,500 unique combinations if the total number of possible combinations for a particular value of k exceeded 1,500. In total, we evaluated 55,723 unique combinations of unperturbable features.

Finally, we also highlight how this experiment demonstrates a type of constraint not covered in the evaluation section: features that the adversary simply does not have control over. Throughout this work, our constraints were defined via the semantics of the domain, i.e. the TCP/IP protocol. However, this experiment to preserve the semantics of the attack also serves as a demonstration of the efficacy of an adversary under this second type of constraints. These results suggest that even if an adversary maintains attack behavior *and* cannot arbitrarily control certain features *and* must obey the TCP/IP protocol, there is still a surprising amount of exploitable attack surface to craft adversarial examples.

6. Discussion

In this section, we describe our thoughts for future work.

Attacking Machine-learning-based NID Systems. In this work, we focused on techniques for attacking NID systems at in the *feature* layer (that is, after packets have been processed by a feature extraction system). Thus, depending on the threat model, an adversary may have to perform an additional series of steps to execute the attack over a network. Specifically, if the adversarial goal is to establish *mistrust* in a machine-learning-based NIDS, then presenting adversarial examples (that is, at the feature layer) is sufficient—by overwhelming the NIDS with adversarial examples, a series of false alarms would be raised, and thus, a real network operator would be unable to distinguish malicious vs benign traffic flows. However, this would require a threat model where an adversary would have direct-access to the input of the NIDS, which may be unreasonable.

A more realistic adversary would be one would who want to execute a malicious payload over the network to another service—therein, the goal of the adversary is to simply have their traffic flow classed as benign, post-feature extraction. Manually curating a traffic flow such that, when extracted, conforms to the specification provided by adversarial examples is a non-trivial (and intractable) process. However, adversaries could leverage the very tools used to generate the attacks used in our evaluated datasets to an astonishingly precise degree. Specifically, the malicious traffic flows in the UNSW-NB15 (of whom features were extracted from), were generated by IXIA Perfect Storm traffic generator [31]. IXIA Perfect Storm can create malicious traffic flows subject to network profile constraints, attack constraints, service constraints, among others [14]. Thus, such traffic generators provide a means for adversaries to specify the constraints presented in an adversarial example, and map them back into real traffic flow.

Attacking Rule-based NID Systems. While machine-learning-based NIDS are posed to be the next generation in intrusion detection systems in commercial settings, traditional rule-based systems, such as Bro [42] and Snort [2], are still relevant today. Here, we posit how adversaries could use adversarial examples to bypass rule-based systems.

In the absence of a machine-learning-based NIDS, adversarial examples can still be generated against rule-based systems, given knowledge about the underlying characteristics of the datasets used to to form the rules. Specifically, for both the NSL-KDD and UNSW-NB15, Snort and Bro logs were used to create the feature vectors after the traffic flows were created. Thus, adversaries would then be equipped with knowledge of the information that would comprise the rules in a Snort or Bro NID system. However, since these rule-based systems are not differentiable (and thus, standard, gradient-based crafting technique would be inapplicable), adversaries would be required to manually construct feature vectors if the adversary knew the rules (as is done to attack non-differentiable learning algorithms, such as Decision Trees [37]), or query-based methods, commonly used in black-box attacks against machine learning classifiers [35].

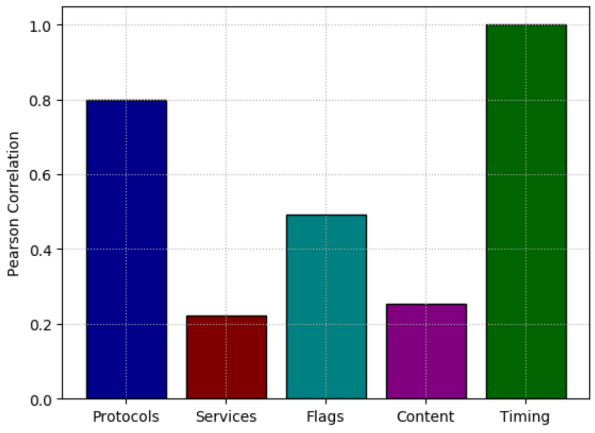


Fig. 5. Normalized mean of summed absolute values of Pearson Correlation Coefficients for five categories of features in the NSL-KDD - Protocols are correlated with the majority of features, closely behind timing-based features, which are highly correlated amongst themselves.

Identifying Primary Features. In this work, we identified primary features manually through not only our understanding of the domain, but observations of the data. We noticed that many features were correlated with the transport layer protocol (as we expected, based on the descriptions of the features). However, we may not always be able to use the descriptions of features to guide us towards identifying primary features. Thus, we hypothesize that primary features could be identified by ranking the features that are most correlated with others. The intuition here is that while secondary features will be highly correlated with primary features, primary features will be highly correlated with the majority of secondary features. We performed a simple experiment where we computed Pearson product-moment correlation coefficients for all features in the NSL-KDD training set. We took the mean of sum of the absolute value of the coefficients for four categories of features as their scores and found protocols to indeed be correlated with the majority of features, as shown in Figure 5. At first, it appeared that timing-based features had a higher correlation than protocols among all features. However, we noticed that the timing-based features were highly correlated *amongst themselves*, which was the largest contributor to their scores (and to no surprise, since many timing-based features are derivatives or direct inverses of one another, e.g., SAME_SRV_RATE and DIFF_SRV_RATE). Thus, if we eliminate such functionally inverse and derivative features, this approach would suggest that primary features could be identified systematically.

Defenses. We also hypothesize that constraints *can* be useful in defending against adversarial examples, even though the network constraints found in the studied datasets were ineffective. We are interested if constraints in other domains, such as malware and spam, can prove to be effective against adversarial example generation algorithms. Intuitively, maintaining the behavior of malware while simultaneously perturbing binary code would be challenging.

In addition, we also observe that our approach for building adversarial sketches can also be used by a defender to assess model vulnerability. In particular, a defender could design a simple mechanism (driven by the perturbation histograms) that reveals universal directions that would make the model vulnerable. The defender can then use this analysis to detect adversarial examples at deployment. However, an adversary could circumvent detection by selectively perturbing features that have less impact. Naturally, this would come at a cost of introducing additional distortion and control over more features, which may be impractical for an adversary.

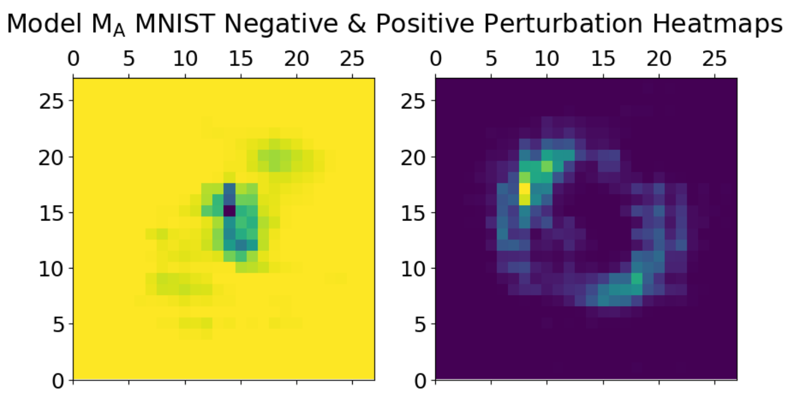
7. Conclusions

This paper investigated the impact of adversarial examples against ML-based network intrusion detection systems through the perspective of traditional adversarial algorithms and universal adversarial perturbations. In addition to this unique investigation into network intrusion detection, we introduced two new algorithms: the ADAPTIVE JSMA, which produces adversarial examples that obey network constraints, and the HISTOGRAM SKETCH GENERATION, which generates adversarial sketches: universal adversarial perturbations that obey network constraints. Our work demonstrates how adversaries can craft permissible adversarial examples against NIDS.

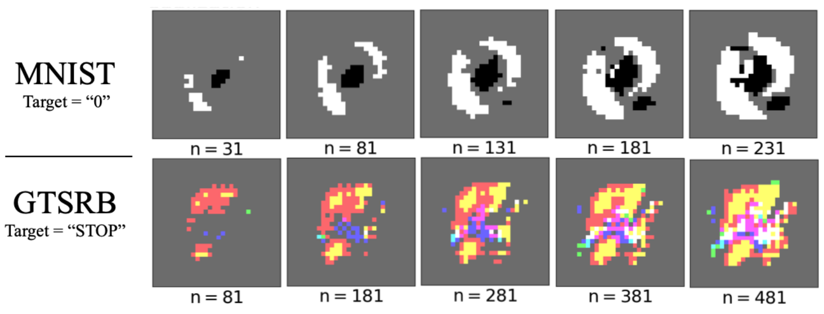
Through our experiments, we observed how biased distributions coupled with low dimensionality can have an impact on model vulnerability, even with network constraints. Furthermore, we showed how when an adversary is constrained to *five random* features, adversarial examples can still be crafted with a $\sim 50\%$ success rate.

Prior to our work, the impact of adversarial learning has been understood in the context of unconstrained domains. We initially hypothesized that the constraints in networks would offer additional resilience to attack algorithms. However, our investigation suggests the inverse. Our two algorithms were able to craft adversarial examples with minimal distortion and with success rates of greater than 95% that also transferred to other models at rates up to 93%.

Appendix. Appendix



(a) Perturbation Histogram produced by the AJSMA for the MNIST model M_A as a 2D Heatmap - The target class (“0”) can be visibly seen by combining the decreasing features plot (left) with the increasing features plot (right).



(b) Sketch Visualization for MNIST and GTSRB for varying values of n - As n increases, the target class visibly forms.

Fig. 6. Perturbation Histogram for MNIST (a) and adversarial sketches for image datasets (b).

	Feature Type			
Protocol	Basic	Content	Timing-based	Host-basted
TCP	duration, service=aol, service=auth, service=bgp, service=courier, service=csnet_ns, service=ctf, service=daytime, service=discard, service=domain, service=echo, service=efs, service=exec, service=finger, service=ftp, service=ftp_data, service=gopher, service=harvest, service=hostnames, service=http, service=http_2784, service=http_443, service=http_8001, service=IRC, service=iso_tsap, service=klogin, service=kshell, service=ldap, service=link, service=login, service=mtp, service=name, service=netbios_dgm, service=netbios_ns, service=netbios_ssn, service=netstat, service=nnsp, service=nntp, service=other, service=pm_dump, service=pop_2, service=pop_3, service=printer, service=private, service=remote_job, service=rje, service=shell, service=smtp, service=sql_net, service=ssh, service=sunrpc, service=supdup, service=systat, service=telnet, service=time, service=uucp, service=uucp_path, service=vmnet, service=whois, service=X11, service=Z39_50, service=34, flag=OTH, flag=REJ, flag=RSTO, flag=RSTOS0, flag=RSTR, flag=S0, flag=S1, flag=S2, flag=S3, flag=SF, flag=SH, src_bytes, dst_bytes, land=1, urgent	hot, num_failed_logins, logged_in=1, num_compromised, root_shell, su_attempted, num_root, num_file_creations, num_shells, num_access_files, is_host_login=1, is_guest_login=1	count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate	dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate
UDP	duration, service=domain_u, service=ntp_u, service=other, service=private, service=tftp_u, flag=SF, wrong_fragment		count, srv_count, serror_rate, rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate	dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_rerror_rate
ICMP	service=eco_i, service=ecr_i, service=red_i, service=tim_i, service=urh_i, service=urp_i, flag=SF, src_bytes, wrong_fragment		count, srv_count, serror_rate, rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate	dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_rerror_rate

Table 5

The constraints extracted from the NSL-KDD - Unlike TCP, UDP and ICMP have limited degrees of freedom.

Learning Technique	Parameters		Test Accuracy			
	Name	Value	NSL-KDD	UNSW-NB15	MNIST	GTSRB
Logistic Regression	PENALTY	l2	74.21%	67.65%	92.02%	84.94%
	C	1.0				
Support Vector Machine	C	1.0	77.31%	69.02%	94.04%	84.3%
	KERNEL	rbf				
	DEGREE	3				
Decision Tree Classifier	CRITERION	gini	74.52%	73.27%	87.73%	57.20%
	MAX_DEPTH	∞				
	MIN_SAMPLES_SPLIT	2				
	MIN_SAMPLES_LEAF	1				
	MAX_FEATURES	∞				
k-Nearest Neighbor	K	5	74.90%	72.20%	96.88%	52.83%
	P	2				

Table 6

Scikit-Learn Model Information

Appendix. References

[1] Abdulsalam Alzahrani and Mohammed Alenazi. 2021. Designing a Network Intrusion Detection System Based on Machine Learning for Software Defined Networks. *Future Internet* 13 (04 2021), 111. https://doi.org/10.3390/fi13050111

[2] Jay Beale. 2004. *Snort 2.1 Intrusion Detection, Second Edition*. Syngress Publishing.

[3] 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 *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 387–402.

[4] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. 2017. Adversarial Patch. *CoRR* abs/1712.09665 (2017). arXiv:1712.09665 http://arxiv.org/abs/1712.09665

[5] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 39–57.

[6] Nicholas Carlini and David A. Wagner. 2018. Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. *CoRR* abs/1801.01944 (2018). arXiv:1801.01944 http://arxiv.org/abs/1801.01944

[7] Graham Cormode. 2011. Sketch techniques for approximate query processing. In *Synposes for Approximate Query Processing: Samples, Histograms, Wavelets and Sketches, Foundations and Trends in Databases. NOW publishers*.

[8] L. Dhanabal and Dr. S. P. Shantharajah. 2015. A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms. *International Journal of Advanced Research and Communication Engineering (IJARCCE)* 4, 6 (2015), 446–452.

[9] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-Box Adversarial Examples for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (Melbourne, Australia). Association for Computational Linguistics, 31–36. http://aclweb.org/anthology/P18-2006

[10] Philippe Flajolet and G. Nigel Martin. 1985. Probabilistic Counting Algorithms for Data Base Applications. *J. Comput. Syst. Sci.* 31, 2 (Sept. 1985), 182–209. https://doi.org/10.1016/0022-0000(85)90041-8

[11] Ian Goodfellow, Nicolas Papernot, Sandy Huang, Peter Duan, Pieter Abbeel, and Jack Clark. 2017. Attacking Machine Learning with Adversarial Examples. https://blog.openai.com/adversarial-example-research/

[12] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. *stat* 1050 (2015), 20.

[13] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*. Springer, 62–79.

[14] W. Haider, J. Hu, J. Slay, B.P. Turnbull, and Y. Xie. 2017. Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. *Journal of Network and Computer Applications* 87 (2017), 185–192. https://doi.org/10.1016/j.jnca.2017.03.018

[15] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: an update. *SIGKDD Explorations* 11, 1 (2009), 10–18.

[16] Jamie Hayes and George Danezis. 2017. Machine Learning as an Adversarial Service: Learning Black-Box Adversarial Examples. *CoRR* abs/1708.05207 (2017). arXiv:1708.05207 http://arxiv.org/abs/1708.05207

[17] Guang-Bin Huang and Haroon A Babri. 1998. Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. *IEEE Transactions on Neural Networks* 9, 1 (1998), 224–229.

[18] Laheeb M Ibrahim, Dujan T Basheer, and Mahmod S Mahmod. 2013. A comparison study for intrusion database (Kdd99, Nsl-Kdd) based on self organization map (SOM) artificial neural network. *Journal of Engineering Science and Technology* 8, 1 (2013), 107–119.

[19] Bhupendra Ingre and Anamika Yadav. 2015. Performance analysis of NSL-KDD dataset using ANN. In *Signal Processing And Communication Engineering Systems (SPACES), 2015 International Conference on*. IEEE, 92–96.

[20] Robin Jia and Percy Liang. 2017. Adversarial Examples for Evaluating Reading Comprehension Systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (Copenhagen, Denmark). Association for Computational Linguistics, 2021–2031. http://aclweb.org/anthology/D17-1215

[21] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14

[22] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli. 2018. Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables. In *2018 26th European Signal Processing Conference (EUSIPCO)*. 533–537. https://doi.org/10.23919/EUSIPCO.2018.8553214

[23] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2017. Adversarial examples in the physical world. In *International Conference on Learning Representations Workshop*. https://openreview.net/pdf?id=S1OufnIlx

[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (Nov 1998), 2278–2324. https://doi.org/10.1109/5.726791

[25] Wenhui Li, Daihui Li, and Shangyou Zeng. 2019. Traffic Sign Recognition with a small convolutional neural network. *IOP Conference Series: Materials Science and Engineering* 688 (12 2019), 044034. https://doi.org/10.1088/1757-899X/688/4/044034

[26] Zilong Lin, Yong Shi, and Zhi Xue. 2021. IDSGAN: Generative Adversarial Networks for Attack Generation against Intrusion Detection. arXiv:1809.02077 [cs.CR]

[27] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJzIBfZAb

[28] Kasongo Mambwe Sydney and Yanxia Sun. 2020. Performance Analysis of Intrusion Detection Systems Using a Feature Selection Method on the UNSW-NB15 Dataset. *Journal Of Big Data* 7 (11 2020). https://doi.org/10.1186/s40537-020-00379-6

[29] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2574–2582. https://doi.org/10.1109/CVPR.2016.282

[30] S. M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. 2017. Universal Adversarial Perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 86–94. https://doi.org/10.1109/CVPR.2017.17

[31] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *Military Communications and Information Systems Conference (MilCIS), 2015*. IEEE, 1–6.

[32] Nour Moustafa and Jill Slay. 2016. The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Information Security Journal: A Global Perspective* 25, 1-3 (2016), 18–31.

[33] Nour Moustafa and Jill Slay. 2017. A hybrid feature selection for network intrusion detection systems: Central points. *arXiv preprint arXiv:1707.05505* (2017).

[34] Nicolas Papernot, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Fartash Faghri, Alexander Matyasko, Karen Hambardzumyan, Yi-Lin Juang, Alexey Kurakin, Ryan Sheatsley, et al. 2016. cleverhans v2. 0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768* (2016).

[35] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks against Machine Learning. arXiv:1602.02697 [cs.CR]

[36] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 372–387.

[37] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. 2016. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. *CoRR* abs/1605.07277 (2016). arXiv:1605.07277 http://arxiv.org/abs/1605.07277

[38] Maria Rigaki and Ahmed Elragal. 2017. Adversarial Deep Learning Against Intrusion Detection Classifiers.

[39] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. 2017. Dynamic Routing Between Capsules. arXiv:1710.09829 [cs.CV]

[40] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. 2016. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1528–1540.

[41] Chris Sinclair, Lyn Pierce, and Sara Matzner. 1999. An application of machine learning to network intrusion detection. In *Computer Security Applications Conference, 1999.(ACSAC'99) Proceedings. 15th Annual*. IEEE, 371–377.

[42] Robin Sommer. 2003. Bro: An Open Source Network Intrusion Detection System. In *DFN-Arbeitstagung über Kommunikationsnetze*.

[43] Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 305–316.

[44] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks* 0 (2012), –. https://doi.org/10.1016/j.neunet.2012.02.016

[45] J. Su, D. V. Vargas, and K. Sakurai. 2019. One Pixel Attack for Fooling Deep Neural Networks. *IEEE Transactions on Evolutionary Computation* (2019), 1–1. https://doi.org/10.1109/TEVC.2019.2890858

[46] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[47] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. 2009. A detailed analysis of the KDD CUP 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*. IEEE, 1–6.

[48] Robert Tibshirani. 1996. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* 58, 1 (1996), 267–288. http://www.jstor.org/stable/2346178

[49] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2017. The Space of Transferable Adversarial Examples. *arXiv preprint arXiv:1704.03453* (2017).

[50] Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin, and Wei-Yang Lin. 2009. Intrusion detection by machine learning: A review. *Expert Systems with Applications* 36, 10 (2009), 11994–12000.

[51] Kaichen Yang, Jianqing Liu, Chi Zhang, and Yuguang Fang. 2018. Adversarial Examples Against the Deep Learning Based Network Intrusion Detection Systems. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*. 559–564. https://doi.org/10.1109/MILCOM.2018.8599759