Rallying Adversarial Techniques against Deep Learning for Network Security

Joseph Clements, Yuzhe Yang, Ankur A. Sharma, Hongxin Hu†, Yingjie Lao
Clemson University, Clemson, South Carolina
{jfcleme, yuzhey, ankurs}@g.clemson.edu, ylao@clemson.edu
†University at Buffalo, Buffalo, New York
hongxinh@buffalo.edu

在这项工作中，我们探索了敌对实体破坏此类漏洞以破坏基于深度学习的 NIDS 系统的潜力

2021 IEEE Symposium Series on Computational Intelligence (SSCI) | 978-1-7281-9048-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/SSCI50451.2021.9660011

***Abstract*—Recent advances in artificial intelligence and the increasing need for robust defensive measures in network security have led to the adoption of deep learning approaches for network intrusion detection systems (NIDS). These methods have achieved superior performance against conventional network attacks, enabling unique and dynamic security systems in real-world applications. Adversarial machine learning, unfortunately, has recently shown that deep learning models are inherently vulnerable to adversarial modifications on their input data. In this work, we explore the potential of adversarial entities to compromise such vulnerabilities to compromise deep learning-based NIDS systems. Specifically, we show that by modifying on average as little as 1.38 of an observed packet's input features, an adversary can generate malicious inputs that effectively fool a target deep learning-based NIDS. Therefore, it is crucial to consider the performance from the conventional network security perspective and the adversarial machine learning domain when designing such systems.**

***Index Terms*—Network Intrusion Detection System, Adversarial Machine Learning, Adversarial Examples, Deep Learning.**

I. Introduction

Mainly attributable to advances in deep learning, the field of artificial intelligence has been growing swiftly in the recent past. Through many examples, it has been witnessed that deep learning systems have the potential to achieve or even surpass human-level performance on specific tasks. Furthermore, these systems are not explicitly given a function to implement but instead can discover hidden rules or patterns that developers may not comprehend. This ability to learn has made deep learning an indispensable tool for advancing the state-of-the-art in multiple fields.

With these remarkable successes, it is unsurprising that deep learning techniques are quickly being adopted in network security for use in intrusion detection [1], malware analysis [2], spam filtering [3], and phishing detection [4]. However, the growing popularity of novel network paradigms (i.e., Internet of Things (IoT) and mobile networks) also brings unique and challenging security requirements. To this end, modern deep learning algorithms can rival traditional approaches, especially in these emerging fields. Recently, the field of deep learning-based network intrusion detection systems (DL-NIDS) has been growing due to the variability and efficiency of the deep learning model. The availability of novel techniques such as recurrent neural networks, semi-supervised learning, and reinforcement learning is allowing DL-NIDS to achieve success in applications that have been traditionally out of the reach of intrusion detection systems [5]–[7].

However, the downside of deep learning is that the high non-linearity seen in these systems limits the ability of developers to guarantee or explain their functionality. This complexity allows for the possibility of unseen security risks. Indeed, many recent works have demonstrated the vulnerability of deep learning to adversarial manipulation [8]–[10]. For example, adversarial examples can completely misclassify a deep learning model by only slightly altering the network input data [11]–[14]. In response to the threat that this form of attack poses to deep learning, multiple potential defenses have arisen [15]–[17]. Despite this, security applications remain vulnerable since it is uncertain which defensive methodologies are most effective in given scenarios.

Therefore, to ensure the defensive capabilities of deep learning-based security systems, these applications should be evaluated against the traditional performance metrics in the target security field and those vulnerabilities from the adversarial deep learning domain. If deployed without understanding these vulnerabilities, a deep learning model could quickly become the most sensitive component of a security system. This paper analyzes adversarial example attacks against current deep learning-based network intrusion detection system (DL-NIDS), demonstrating the real-world vulnerability of such systems. Specifically, we investigate the security of a DL-NIDS that has been used for security analysis, Kitsune. This system offers a similar level of defensive capability as traditional intrusion detection systems while requiring a lower overhead [18]. We evaluate the DL-NIDS from two perspectives: 1) the ability to defend from malicious network attacks and 2) the robustness against adversarial examples. Our experimental evaluations find that this model is vulnerable to adversarial manipulation through its deep learning model. Thus, we argue that the benefit of such systems should be evaluated by their ability to defend from traditional attacks and the vulnerability of the deep learning model to manipulation.

In the remainder of the paper, we first introduce the basics of deep learning and its use in intrusion detection systems and the state-of-the-art in deep adversarial learning. We decompose the target DL-NIDS, Kitsune, in Section III. Then, we briefly outline our experimental setup in Section IV. In Sections V and VI, we evaluate the DL-NIDS from the perspectives of

978-1-7281-9048-8/21/$31.00 ©2021 IEEE

network security and deep adversarial learning, respectively. Finally, Section VII concludes the paper.

II. Background

A. Deep Learning based Network Intrusion Detection Systems

In recent years, the increasing frequency and size of cyber-attacks in recent years [19] have made network intrusion detection systems (NIDS) a critical component in network security. An example of a network intrusion detection system is shown in Figure 1. The intrusion detection system essentially acts as a gatekeeper at the target node, which activates a firewall or alerts a host device when malicious network traffic is detected. Unfortunately, while these systems can effectively defend the entry point, much of the network remains unprotected. In other words, attacks that remain internal to the network are often difficult to detect by the traditional intrusion detection systems [18].

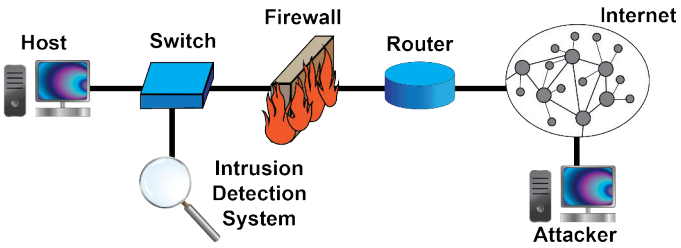

Fig. 1. An intrusion detection system positioned to defend a host device from abnormal network traffic.

Deploying an intrusion detection system at multiple nodes distributed throughout the network can fill this hole to further secure networks. However, a significant drawback of the traditional rule-based approach is that each intrusion detection system must be explicitly programmed to follow a set of rules. This process also generates potentially long lists of rules that need to be stored locally to access intrusion detection systems. Furthermore, any changes in a network node might potentially lead to an update for the entire network. To this end, DL-NIDS have the potential to overcome this weakness as they can generalize the defense by capturing the distribution of typical network traffic instead of being explicitly programmed [18], [20]–[22]. In addition, these methods do not require large lookup tables, which could also reduce the implementation cost.

B. Adversarial Example Generation

A major focus of adversarial deep learning is the adversarial example generation, which attempts to find input samples by slightly perturbing the original benign data to yield different classifications. Formally, the adversarial example generation process can be expressed by [11]:

$$\begin{aligned} \text{minimize} \quad & \mathcal{D}(\vec{x}, \vec{x}+\vec{\delta}) \\ \text{such that} \quad & \mathcal{C}(\vec{x}+\vec{\delta}, \vec{t}) \\ & \vec{x}+\vec{\delta} \in \mathbb{X} \end{aligned} \tag{1}$$

Where $\vec{x}$ is the model's original primary input, $\vec{\delta}$ is a perturbation on $\vec{x}$ to achieve the desired adversarial behavior, and $\mathbb{X}$ defines a bounded region of the valid input values. $\mathcal{D}(\cdot)$ is a distance metric that limits δ, while $\mathcal{C}(\cdot)$ is a constraint that defines the goal of the attack. Two commonly used constraint functions are $F(\vec{x}) = \vec{t}$ and $F(\vec{x}) \neq \vec{t}$. The first defines a targeted attack in which the adversarial goal is to force the network output, $F(\vec{x})$, to a specific output, $\vec{t}$. The second defines the untargeted scenario where the adversarial goal is for the network to produce any output except $\vec{t}$. The choice of $\mathcal{D}(\cdot)$ also greatly affects the outcome of the attack. In the existing works, L_P norms (i.e., L_0, L_1, L_2, and L_∞) are often used due to their mathematical significance and correlation with perceptual distance in image or video recognition. Recently, new distance metrics are being explored with the recent works such as spatially transformed adversarial examples [23].

Many algorithms for generating adversarial examples utilizing various $C(\cdot)$, $D(\cdot)$, and optimization approaches have been developed in the literature. For example, one of the earliest adversarial example algorithms, Fast Gradient Sign Method (FGSM), perturbs every element of the input in the direction of its gradient by a fixed size [12]. While this method produced quick results, the Basic Iterative Method (BIM) can significantly decrease the perturbation, requiring a longer time to run [16]. Furthermore, adversarial example generation algorithms continue to grow more sophisticated as novel attacks build on the foundation of existing works. An example of this is the elastic net method (ENM) which adds an L_1 regularization term and the iterative shrinkage-thresholding algorithm to Carlini and Wagner's attack [14]. Moreover, adversarial examples are expanding out from image processing into alternate fields where they continue to inhibit the functionality of deep learning models [24]–[26]. The effort to draw researcher awareness to the subject has even lead to the generation of competitions in which contestants attempt to produce and defend neural networks from this adversarial example [27], [28].

C. Robustness against Adversarial Examples

Some researchers believe that the vulnerability of deep learning models to adversarial examples is evidence of a pervasive lack of robustness rather than simply an inability to secure these models [29]–[31]. As such, defenses attempt to bolster the deep learning model's robustness by using either reactive or proactive methods [32]. Defensive distillation and adversarial training are two proactive defenses, which improve a neural network's robustness by retraining the network weights to smooth the classification space [15], [16]. A recent example of a reactive defense is, PixelDefend, which attempts to perturb adversarial example input back to the region of inputs space that is correctly handled by the network [17].

When deep learning is powering security applications, the robustness of the model is even more critical. The field of malware classification is a prime example as deep learning models have been shown to perform superbly in this area in multiple

implementations and scenarios [33]–[36]. Unfortunately, when adversarial examples are presented to these systems, the lack of robustness in the deep learning model often allows an attacker to bypass these security measures [37], [38]. Despite this vulnerability, deep learning is a prime candidate for security implementations when traditional defenses' resource demands or static nature inhibit their practicality. Thus, as deep learning continues to develop into network intrusion detection, the robustness of such systems should be thoroughly studied. To this end, researchers are continuing to develop guidelines and frameworks to aid in ensuring the robustness of machine learning systems against adversarial manipulations [39], [40].

III. Evaluated Network

This section presents a brief overview of the network intrusion detection system and then analyzes Kitsune's deep learning model, KitNET, in more detail.

A. Kitsune Overview

DL-NIDS, Kitsune，由数据包捕获器、数据包解析器、特征提取器、特征映射器和异常检测器组成 [18]。Packet Capturer 和 Packet Parser 是 NIDS 的标准组件，它们将解析后的数据包和元信息（例如，传输通道、网络抖动、捕获时间）转发给 Feature Extractor。然后，特征提取器生成一个包含 100 多个统计信息的向量，它定义了活动通道的数据包和当前状态。Feature Mapper 将这些特征聚类成子集，然后输入到异常检测器中，异常检测器包含深度学习模型 KitNET

The DL-NIDS, Kitsune, is composed of Packet Capturer, Packet Parser, Feature Extractor, Feature Mapper, and Anomaly Detector [18]. The Packet Capturer and Packet Parser are standard components of NIDS, which forward the parsed packet and meta-information (e.g., transmission channel, network jitter, capture time) to the Feature Extractor. Then, the Feature Extractor generates a vector of over 100 statistics which defines the packet and current state of the active channel. The Feature Mapper clusters these features into subsets fed into the Anomaly Detector, which houses the deep learning model, KitNET.

The Kitsune DL-NIDS is specifically targeted at being a lightweight intrusion detection system deployed on network switches in the IoT settings. Thus, each implementation of Kitsune should be tailored to the network node that it defends. This goal is achieved by using an unsupervised online learning approach that allows the DL-NIDS to dynamically update in response to the traffic at the target network node. The algorithm assumes that all real-time transmissions during the training stage are legitimate and thus learns a benign data distribution. For inference, it analyzes the incoming transmissions to determine if it resembles the learned distribution.

定制

B. KitNET

KitNET, Kitsune's deep-learning backbone, consists of an ensemble layer and an output layer. The ensemble layer includes multiple autoencoders, each working on a single cluster of inputs provided from the Feature Mapper. The output scores of these autoencoders are normalized before being passed to an aggregate autoencoder in the output layer, whose score is used to assess the security of the network traffic data.

1) The Autoencoders: The fundamental building block of KitNET is an autoencoder, a neural network that reduces an input down to a base representation before reconstructing to the same input dimension from that representation. The autoencoders in KitNET are trained to capture the properties of typical network traffic correctly. The number of hidden neurons inside an autoencoder is limited so the network can learn a compact representation.

KitNET employs a root-mean-squared-error (RMSE) function on each autoencoder as the performance criteria. The score generated by each autoencoder block is given by:

$$s(\vec{x}) = RMSE(\vec{x}, F(\vec{x})) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - F(\vec{x})_i)^2}{n}} \quad (2)$$

where n is the number of inputs. Because the model was trained to reproduce instances from X, a low score indicates the input resembles the normal distribution well.

2) The Normalizers: Another component used by Kitsune is the normalizers, appearing both before entering KitNET and before the aggregate autoencoder. These normalizers implement the standard function:

$$norm(x_i) = \frac{x_i - min_i}{Max_i - min_i} \quad (3)$$

which linearly scales minimum and maximum input values to 0 and 1, respectively. In Kitsune's training, the value of Max_i and min_i respectively take on the maximum and minimum input values seen by the $x_{i_{th}}$ element during training.

C. Classifying the Output

The primary output of KitNET is the RMSE score, S, produced by the aggregate autoencoder. It should be noted that the scores produced by KitNET are numerical values rather than a probability distribution or logits like in standard deep learning classifiers. Kitsune utilizes a classification scheme which triggers an alarm under the condition: $S \geq \phi\beta$, where ϕ is the highest value of S recorded during training and β is a constant used to find a trade-off between the number of false positives and negatives. The authors limit the value of β to be greater than or equal to 1.0 to assure a 100% training accuracy (i.e., all the training data are considered benign).

IV. Experimental Setup

In this section, we briefly describe our experimental setup and the necessary modifications to the KitNET.

A. Implementing KitNET

Kitsune原来是用C++，后来转成了TensorFlow

In order to perform adversarial machine learning, the original C++ version of Kitsune was reproduced in TensorFlow [41]. The TensorFlow model was tested and evaluated similarly to the C++ implementation with an average deviation on the outputs of 5.71×10^{-7} from the original model. We then utilized the Cleverhans [42], an adversarial machine learning library that is produced and maintained by domain experts, to mount different adversarial example generation algorithms on the Kitsune. We also used the same Mirai dataset as in [18].

B. Modifications to the Model

Our implementation of KitNET moves the classification mechanism into the model by adding a final layer at the output, as expressed in Equation 4.

$$C(\vec{x}) = \begin{bmatrix} \text{benign} \\ \text{malicious} \end{bmatrix} = S(\vec{x}) \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 0 \\ 2T \end{bmatrix} \quad (4)$$

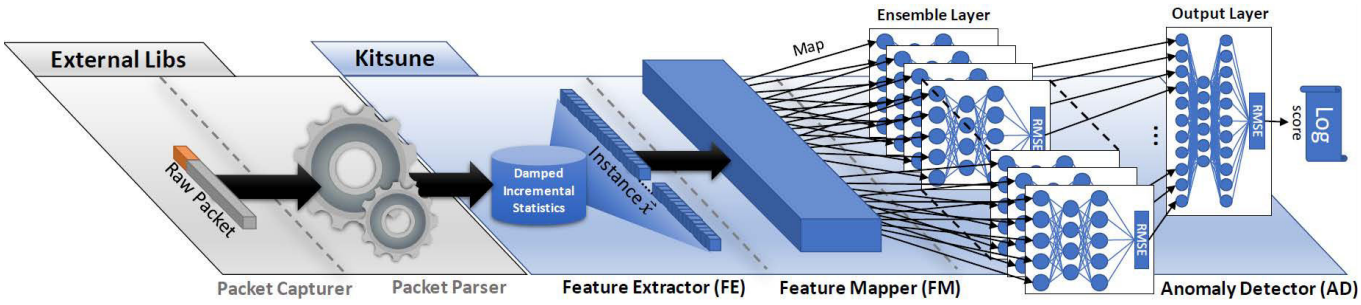

Fig. 2. A graphical representation of Kitsune [18].

This allows the deep learning model to produce the classification result based on a threshold, T. Effectively, this alteration moves the original classification scheme into KitNET itself when $T = \phi\beta$, transforming the model from a regression model into a classifier.

As adversarial examples target deep learning models, we isolate KitNET from Kitsune when performing our attacks. In a real-world attack on Kitsune, the adversary must circumvent, or surmount, the Feature Extractor to induce perturbations on KitNET's input. However, understanding the Feature Extractor makes it feasible for the adversary to craft network traffics to generate essential features. Thus, in our experiments, we focus on evaluating the security of KitNET from the normalized feature space.

V. Evaluation from the Network Security Perspective

A DL-NIDS must be evaluated from both the network security and adversarial machine learning aspects to understand its defensive capabilities fully. In the domain of intrusion detection, the ability to distinguish malicious network traffics from benign traffics is the primary performance metric. In this section, we evaluate the classification accuracy of the Kitsune.

Kitsune's developers evaluate the DL-NIDS against a series of attacks in a variety of networks [18]. In our implementation, the accuracy of Kitsune is highly dependent on the threshold, T. This value defines the decision boundary, which makes it a critical parameter when deploying the model. We evaluate the KitNET by assuming that the threshold is not predefined but trained as an end-to-end deep learning system. In addition, this analysis also indicates how the threshold correlates with the perturbation required in adversarial machine learning.

To assess the performance of a given threshold value, we consider the following two metrics:

1) **False Positives:** The percentage of benign inputs that are incorrectly classified as malicious.
2) **False Negatives:** The percentage of malicious data that are incorrectly classified as benign.

On the one hand, the rate of false positives accounts for the reliability of a network. On the other hand, the rate of false negatives is closely associated with the intrusion detection system's effectiveness. Therefore, both rates should be minimized in an ideal situation. However, in the setting of Kitsune, the value of T acts as a trade-off between false positive rate and false-negative rate.

We investigated the whole functional range of possible thresholds in this analysis, i.e., from the minimum score of 0 to 20, which leads to 100% false negatives on the given dataset. Figure 3(a) plots the two metrics as well as the accuracy of the DL-NIDS.

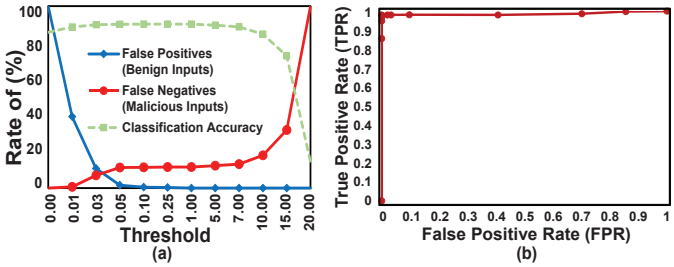

Fig. 3. The percentage of misclassified benign and malicious inputs for chosen threshold values (a). A receiver operating characteristic (ROC) curve for Kitsune (b).

It can be seen that the rates of false positives and false negatives remain almost unchanged in the middle range. Furthermore, it can also be observed that if we want to minimize one of the rates, the other rate will increase significantly. Finally, the accuracy is also essentially unchanged for threshold values below 7, which can partially contribute to the imbalance of the dataset (i.e., most of the data belong to the benign class). Therefore, a threshold between 0.05 and 1 would be appropriate for this scheme. The effectiveness of Kitsune at separating the Mirai dataset is further demonstrated by the ROC curve in Figure 3(b).

VI. Evaluation against Adversarial Machine Learning

This section continues the evaluation of Kitsune through an empirical analysis of its robustness against adversarial examples.

A. *Adversarial Example Generation Methods*

Intelligent and adaptive adversaries will exploit the vulnerability of the machine learning models against novel DL-NIDS by using techniques such as adversarial examples and poisoning attacks. There are mainly two attacking objectives

完整性和可用性违规

in adversarial machine learning, namely, integrity and availability violations. In this setting, integrity violations attempt to generate malicious traffic, which evades detection (produce a false negative), while availability violations attempt to make benign traffic appear malicious (produce a false positive) [43]. However, adversarial examples attempt to achieve a misclassification with perturbations as small as possible.

Another concern in performing these attacks is that the network data are fundamentally distinct from images, usually used in conventional adversarial machine learning. An adversarial example in the image domain is an image perceived to be the same by human observers but differently by the model. The L_P norm between the two images exemplifies visual distance and can be used as the distance metric. However, this definition fails in network security as observing network traffic at the bit-level is not generally practical. Therefore, the semantic understanding of these attacks in this setting is remarkably different.

也就是说，对NIDS的对抗样本不强调语义上的距离问题

然而，这个定义在网络安全方面是失败的，因为在比特级观察网络流量通常是不切实际的。因此，在此设置中对这些攻击的语义理解明显不同。

One potential definition for adversarial examples in this scenario, which is facilitated by the architecture of Kitsune, is to use the extracted features generated by the model as an indication of the observable difference. Thus, we adopt the L_P distance on the feature space between the original input and the perturbed input as the distance metric. In particular, the L_0 norm correlates to altering a small number of extracted features, which might be a better metric than other L_P norms.

在这种情况下，对抗性示例的一个潜在定义（由 Kitsune 的架构提供便利）是使用模型生成的提取特征作为可观察差异的指示。因此，我们采用原始输入和扰动输入之间的特征空间上的 LP 距离作为距离度量。

Many methods of generating adversarial examples have been developed. With each thrives in different settings, we attempt to generate a broad comparison of adversarial examples with different distance metrics in the network security domain. We evaluate the robustness of the KitNET against the following algorithms:

- **Fast Gradient Sign Method (FGSM):** This method optimizes over the L_∞ norm (i.e., reduces the maximum perturbation on any input feature) by taking a single step to each element of $\vec{x}$ in the direction opposite the gradient [12].
- **Jacobian Base Saliency Map (JSMA):** This attack minimizes the L_0 norm by iteratively calculating a saliency map and then perturbing the feature that will have the highest effect [13].
- **Carlini and Wagner (C&W):** Carlini and Wagner's adversarial framework, as discussed earlier, can either minimize the L_2, L_0 or L_∞ distance metric [11]. Our experiments utilize the L_2 norm to reduce the Euclidean distance between the vectors through an iterative method.
- **Elastic Net Method (ENM):** Elastic net attacks are novel algorithms that limit the total absolute perturbation across the input space, i.e., the L_1 norm. ENM produces the adversarial examples by expanding an iterative L_2 attack with an L_1 regularizer [14].

B. Experimental Results

We conduct our experiments on both integrity and availability violations. Integrity violation attacks are performed on the benign inputs with a threshold of $s = 1.0$. The experimental results are presented in Table I. For comparison between different algorithms, the common L_P distance metrics are all presented. Each attack was conducted on the same 1000 random benign samples from the dataset.

假良性，对恶意流量样本扰动，使之逃逸检测

TABLE I
INTEGRITY ATTACKS ON KITNET

		L_P Distances			
Algorithm	Success (%)	L_0	L_1	L_2	L_∞
FGSM	100	100	108	10.8	1.8
JSMA	100	2.33	10.73	6.97	4.87
C&W	100	100	7.44	3.61	3.49
ENM	100	1.21	4.94	4.64	4.49

Availability attacks are also performed using the same threshold of $s = 1.0$. 1000 input vectors that yield the closest output scores to the threshold were selected. The results are summarized in Table II. As the normalizers were only trained on benign inputs, many malicious inputs would be normalized outside the typical range between 0 and 1.

假阳性，对干净流量样本扰动，使之被误判恶意

TABLE II
AVAILABILITY ATTACKS ON KITNET

		L_P Distances			
Algorithm	Success (%)	L_0	L_1	L_2	L_∞
FGSM	4	100	78.00	7.79	0.78
JSMA	0	–	–	–	–
C&W	100	100	22.00	8.50	5.61
ENM	100	8.74	21.7123	8.14	3.60

C. Analysis and Discussion

By comparing Table I and Table II, it can be seen that the integrity attacks, in general, perform much better than the availability attacks. For instance, adversarial examples are rarely generated in the FGSM and JSMA availability attacks. Additionally, the perturbations produced by the availability attacks are all larger than their integrity counterparts. A potential cause for the difficulty is the disjoint nature between the benign and malicious input data, as exhibited by the clipping of the normalized inputs, in conjunction with a boundary decision (i.e., the threshold T) that is much closer to the benign input data.

Among these four methods, the earlier algorithms, i.e., the FGSM and JSMA, perform worse than the C&W and ENM attacks. As we mentioned above, especially in the availability attacks, the success rates of these attacks are significantly low. This result is expected since the more advanced iterative C&W and ENM algorithms can search a larger adversarial space than the FGSM and JSMA.

A final observation is that ENM is very effective in these attacks. Even though this attack is optimized for the L_1 norm, its generated adversarial examples simultaneously yield minimal values for the other norms. Specifically, the L_0 perturbations produced were even better than those produced by JSMA. As stated above, the L_0 norm seems to be the most appropriate norm among these four L_p norms in the setting of

network security, as it signifies altering a minimized number of extracted features from the network traffic. Thus, ENM can be implemented against the Kitsune to generate adversarial examples to fool the detection system while requiring minimal perturbations.

We note that the above attacks were produced with an adaptive step size random search of the parameters of each method. In practice, adversaries may use such a naive approach to determine effect attack algorithms. Then, utilize more robust optimization algorithms, such as Bayesian or gradient descent optimization, with the indicated attack algorithms to produce a superior result.

D. Optimizing ENM

Since ENM has been demonstrated to be very successful in our experiments, we next focus on optimizing the ENM attack on Kitsune in our setting. The CleverHans implementation uses a simple gradient descent optimizer to minimize the function:

$$c \cdot \max\{F(\vec{x})_j - Y \ , \ 0\} + \beta||\vec{x} - \vec{x_0}||_1 + ||\vec{x} - \vec{x_0}||_2 \quad (5)$$

where $F(\cdot)_j$ is the logit output of the target classifier, Y is the target logit output (i.e., the output which produces the desired violation), and $\vec{x_0}$ is the original network input. It can be seen that there are two regularization parameters, c and β. These parameters determine the contribution of the different metrics to the attack algorithm. For example, a very large c effectively increases the attack's ability to converge to a successful attack. The large contribution of the constraint terms also potentially overshadows the distance metrics, effectively diminishing the attack's ability to minimize the perturbation. The focus of this optimization is to determine optimal regularization terms to produce effective attacks on KitNET.

The ENM algorithm has several other hyper-parameters, including the learning rate, maximum gradient descent steps, and targeted confidence level. These parameters are standard in adversarial example attacks; these parameters are set to the constant values of 0.05, 1000, and 0, respectively. An optimization scheme included in the ENM algorithm aids in producing optimal results by altering c. It decreases the parameter N-times, only retaining the successful attack, which produces the lowest perturbation. This feature is disabled by setting $N = 0$, ensuring that it does not alter optimization results. Therefore, the results of the optimization could be further improved by enabling this functionality.

The parameter, c, determines the contribution of the adversarial misclassification objective at the cost of diminishing the two L_P normalization terms. Thus, it can be logically determined that the optimal value of c is that value that achieves the demanded success rate while remaining as small as possible. We evaluate a wide range of c values for $\beta = 1$, as shown in Figure 4. We find $c = 450$ optimal, which achieves a 100% success rate with a relatively small perturbation. It can also be observed from Figure 4 that the resultant L_1 distance does not directly correlate to the selection of c. We also tried to increase the value of c into the thousands; interestingly, the L_P distances still only changed very slightly.

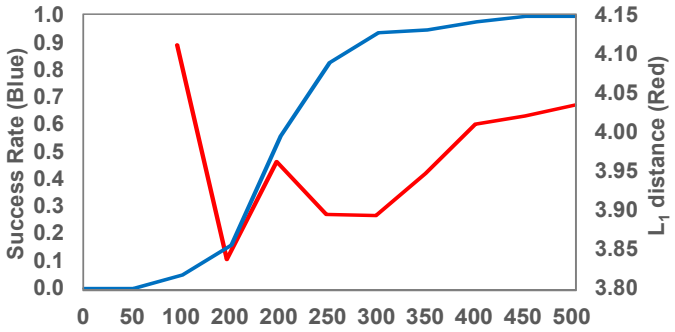

Fig. 4. The success rate (blue) and average L_1-distance (red) of adversarial examples with respect to the regularization parameter, c, used for the attack.

On the other hand, the choice of β significantly affects the L_P distances. We now optimize the produced perturbation through varying the parameter β for $c = 450$. The results are summarized in Table III. It can be seen that the success rate will drop as the increase of c, after the second term of Equation 5 begins to overpower the loss function associated with c.

TABLE III
THE PERTURBATIONS PRODUCED WITH RESPECT TO β.

		L_P Distances			
β	Success (%)	L_0	L_1	L_2	L_∞
1×10^{-5}	100	96.61	5.9518	3.6378	3.5163
1×10^{-4}	100	78.46	5.7574	3.6388	3.516
1×10^{-3}	100	33.34	5.0577	3.6435	3.5268
1×10^{-2}	100	5.51	5.1722	3.7658	3.3129
1×10^{-1}	100	1.09	3.8624	3.7277	3.6450
1×10^{0}	100	1.01	4.0347	4.0158	4.0044
2×10^{1}	0.84	1.00	4.1350	4.1350	4.1350
5×10^{1}	0.08	1.00	4.2054	4.2054	4.2054
1×10^{2}	0	-	-	-	-

Summary: It can be concluded that adversarial machine learning can be a real threat against DL-NIDS. Therefore, when moving intrusion detection towards the profound learning realm, it is critical to evaluate the security of a DL-NIDS against both adversarial attacks in the conventional network and the machine learning domains.

VII. CONCLUSIONS AND FUTURE DIRECTIONS

This paper has demonstrated the vulnerability of DL-NIDS to well-crafted attacks from the domain of adversarial machine learning. This vulnerability is present in deep learning-based systems even when the model achieves high accuracy for classifying benign and malicious network traffic. Therefore, researchers must take steps to verify the security of deep learning models in security-critical applications to ensure they do not impose additional risks; otherwise, it will defeat the purpose of using deep learning techniques to protect networks.

The existence of the Feature Extractor and the Packet Parser signifies that the Kitsune is at least partially utilizing domain knowledge of network traffic to generate its classification. Their applications strive to be as data-driven as possible to get the most benefit from deep learning models (i.e., they require little to no human knowledge to generate a function mapping). Thus, despite the current success of Kitsune and other DL-NIDS, as the field continues to develop, DL-NIDS will attempt directly converting network traffic to a classification utilizing end-to-end deep learning models. Furthermore, the human knowledge currently being used by modern DL-NIDS implies that to increase the probability of a successful attack, an adversary should understand this knowledge. Thus, as DL-NIDS continues to develop, evaluating the model against adversarial machine learning techniques becomes even more critical as attacks will no longer require this additional knowledge when targeting the system.

This work assumes that the adversary has direct knowledge of the target DL-NIDS, allowing them to directly generate inputs for the deep learning model. A potential drawback of this assumption is that the perturbation requires to generate the adversarial examples does not directly correlate to the alteration on the network. Additionally, it does not account for the effect of that change on the network traffic on the host device. Future works will address this gap between the adversarial input to the deep learning model and the network traffic.

ACKNOWLEDGEMENT

This work is partially supported by the National Science Foundation award 2047384.

REFERENCES

[1] R. N. Thomas and R. Gupta, "Design and development of an efficient network intrusion detection system using machine learning techniques," *Wireless Communications & Mobile Computing*, 2021.

[2] Z. Moti, S. Hashemi, H. Karimipour, A. Dehghantanha, A. N. Jahromi, L. Abdi, and F. Alavi, "Generative adversarial network to detect unseen internet of things malware," *Ad Hoc Networks*, 2021.

[3] Y. Kontsewaya, E. Antonov, and A. Artamonov, "Evaluating the effectiveness of machine learning methods for spam detection," EasyChair, Tech. Rep., 2021.

[4] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from urls," *Expert Systems with Applications*, vol. 117, pp. 345–357, 2019.

[5] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *Ieee Access*, vol. 5, pp. 21 954–21 961, 2017.

[6] R. A. R. Ashfaq, X.-Z. Wang, J. Z. Huang, H. Abbas, and Y.-L. He, "Fuzziness based semi-supervised learning approach for intrusion detection system," *Information Sciences—Informatics and Computer Science, Intelligent Systems, Applications: An International Journal*, vol. 378, pp. 484–497, 2017.

[7] R. Vishwakarma and A. K. Jain, "A honeypot with machine learning based detection framework for defending iot based botnet ddos attacks," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, 2019, pp. 1019–1024.

[8] W. Li, J. Yu, X. Ning, P. Wang, Q. Wei, Y. Wang, and H. Yang, "Hu-Fu: Hardware and Software Collaborative Attack Framework against Neural Networks," *International Symposium on Very Large Scale Integration (ISVLSI)*, 2018.

[9] Y. Liu, Y. Xie, and A. Srivastava, "Neural Trojans," *arXiv preprint arXiv:1710.00942*, 2017.

[10] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[11] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.

[12] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *International Conference on Learning Representations (ICLR)*, 2015.

[13] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016, pp. 372–387.

[14] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh, "Ead: elastic-net attacks to deep neural networks via adversarial examples," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[15] Z. Liu, Q. Liu, T. Liu, N. Xu, X. Lin, Y. Wang, and W. Wen, "Feature distillation: Dnn-oriented jpeg compression against adversarial examples," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 860–868.

[16] R. Duan, X. Ma, Y. Wang, J. Bailey, A. K. Qin, and Y. Yang, "Adversarial camouflage: Hiding physical-world attacks with natural styles," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1000–1008.

[17] X. Yin, S. Kolouri, and G. K. Rohde, "Gat: Generative adversarial training for adversarial example detection and robust classification," in *International Conference on Learning Representations*, 2019.

[18] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[19] K. Huang, M. Siegel, and S. Madnick, "Systematically understanding the cyber attack business: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.

[20] H. M. Song, J. Woo, and H. K. Kim, "In-vehicle network intrusion detection using deep convolutional neural network," *Vehicular Communications*, vol. 21, p. 100198, 2020.

[21] M. Islabudeen and M. K. Devi, "A smart approach for intrusion detection and prevention system in mobile ad hoc networks against security attacks," *Wireless Personal Communications*, vol. 112, no. 1, pp. 193–224, 2020.

[22] M. Almiani, A. AbuGhazleh, A. Al-Rahayfeh, S. Atiewi, and A. Razaque, "Deep recurrent neural network for iot intrusion detection system," *Simulation Modelling Practice and Theory*, vol. 101, p. 102031, 2020.

[23] C. Xiao, J.-Y. Zhu, B. Li, W. He, M. Liu, and D. Song, "Spatially transformed adversarial examples," *International Conference on Learning Representations (ICLR)*, 2018.

[24] N. Carlini and D. Wagner, "Audio adversarial examples: Targeted attacks on speech-to-text," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 1–7.

[25] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," *arXiv preprint arXiv:1702.02284*, 2017.

[26] J. Kos, I. Fischer, and D. Song, "Adversarial examples for generative models," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 36–42.

[27] W. Brendel, J. Rauber, A. Kurakin, N. Papernot, B. Veliqi, M. Salathé, S. P. Mohanty, and M. Bethge, "Adversarial vision challenge," *arXiv preprint arXiv:1808.01976*, 2018.

[28] A. Kurakin, I. Goodfellow, S. Bengio, Y. Dong, F. Liao, M. Liang, T. Pang, J. Zhu, X. Hu, C. Xie *et al.*, "Adversarial attacks and defences competition," in *The NIPS'17 Competition: Building Intelligent Systems*. Springer, 2018, pp. 195–231.

[29] N. Ford, J. Gilmer, N. Carlini, and D. Cubuk, "Adversarial examples are a natural consequence of test error in noise," *arXiv preprint arXiv:1901.10513*, 2019.

[30] S. H. Silva and P. Najafirad, "Opportunities and challenges in deep learning adversarial robustness: A survey," *arXiv preprint arXiv:2007.00753*, 2020.

[31] J. Gilmer, L. Metz, F. Faghri, S. S. Schoenholz, M. Raghu, M. Wattenberg, and I. Goodfellow, "Adversarial spheres," *arXiv preprint arXiv:1801.02774*, 2018.

[32] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, 2019.
[33] J. Gu, B. Sun, X. Du, J. Wang, Y. Zhuang, and Z. Wang, "Consortium blockchain-based malware detection in mobile devices," *IEEE Access*, vol. 6, pp. 12 118–12 128, 2018.
[34] V. Niveditha, T. Ananthan, S. Amudha, D. Sam, and S. Srinidhi, "Detect and classify zero day malware efficiently in big data platform," *International Journal of Advanced Science and Technology*, vol. 29, no. 4s, pp. 1947–1954, 2020.
[35] H. Naeem, F. Ullah, M. R. Naeem, S. Khalid, D. Vasan, S. Jabbar, and S. Saeed, "Malware detection in industrial internet of things based on hybrid image visualization and deep learning model," *Ad Hoc Networks*, vol. 105, p. 102154, 2020.
[36] R. Feng, S. Chen, X. Xie, G. Meng, S.-W. Lin, and Y. Liu, "A performance-sensitive malware detection system using deep learning on mobile devices," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1563–1578, 2020.
[37] H. Li, S. Zhou, W. Yuan, J. Li, and H. Leung, "Adversarial-example attacks toward android malware detection system," *IEEE Systems Journal*, vol. 14, no. 1, pp. 653–656, 2019.
[38] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 2018, pp. 533–537.
[39] J. Gilmer, R. P. Adams, I. Goodfellow, D. Andersen, and G. E. Dahl, "Motivating the rules of the game for adversarial example research," *arXiv preprint arXiv:1807.06732*, 2018.
[40] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, and A. Madry, "On evaluating adversarial robustness," *arXiv preprint arXiv:1902.06705*, 2019.
[41] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.
[42] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, and R. Long, "Technical report on the cleverhans v2.1.0 adversarial examples library," *arXiv preprint arXiv:1610.00768*, 2018.
[43] G. Apruzzese, M. Colajanni, L. Ferretti, A. Guido, and M. Marchetti, "On the effectiveness of machine and deep learning for cyber security," in *2018 10th International Conference on Cyber Conflict (CyCon)*. IEEE, 2018, pp. 371–390.