Examining the Robustness of Learning-Based DDoS Detection in Software Defined Networks

Ahmed Abusnaina[†⋆], Aminollah Khormali[†⋆], DaeHun Nyang[‡], Murat Yuksel[†], Aziz Mohaisen[†]
[†]University of Central Florida [‡]Inha University [⋆]Contributed equally (ordered alphabetically)

***Abstract*—With the rapid development of Software-Defined Networking (SDN) advocating a centralized view of networks, efficient and reliable Distributed Denial of Service (DDoS) defenses are necessary to protect the centralized SDN controller. Recently, an amalgamation of work has realized such defenses using Deep Learning (DL) based algorithms. Although DL-based algorithms are generally prone to adversarial learning attacks, the extent to which those attacks are applicable to DDoS defenses in SDN is unexamined. In this work, we explore the robustness of DL-based DDoS defenses in SDN against adversarial learning attacks. First, we investigate generic off-the-shelf adversarial attacks to test the robustness of DDoS defenses in SDN, and demonstrate that while they lead to misclassification, these attacks do not preserve the characteristics of flows. As a result, we propose Flow-Merge for realistic adversarial flows while achieving a high evasion rate, with both targeted and untargeted misclassification attacks. The proposed Flow-Merge is able to force the DL-based DDoS defenses to misclassify 100% of benign flows as malicious, while preserving original characteristics of flows. Using state-of-the-art defenses, we show that the adversarial flows generated using Flow-Merge are difficult to detect, with only 49.31% detection rate when using adversarial training.**

***Keywords*-Intrusion Detection Systems; Deep Learning; Adversarial Machine Learning; Software Defined Networking**

I. Introduction

Software Defined Networking (SDN) overcomes scalability challenges in network management by a centralized view of a network with many components. A programmable controller in SDN can see all switches and endpoints in a network and manage flows between them, providing a better and easier network monitoring and enhancing security compared to traditional networks [1], [2]. On the other hand, it has been shown that SDNs are vulnerable to Distributed Denial of Service (DDoS) attacks, which target the centralized controller [3], [4]. Those attacks flood services with malicious or undesirable traffic, disallowing them from processing legitimate requests

To address those attacks, several studies develop anomaly detectors for SDN using Machine Learning (ML) techniques, including Deep Learning (DL). Niyaz *et al.* [5] proposed a DL-based DDoS detection system for SDN. Abubaker *et al.* [6] introduced a flow-based IDS for SDN using machine learning. Tang *et al.* [7] proposed a gated recurrent unit RNN-based IDS over SDN-based networks. Ahmed *et al.* [8] introduced SDN-based networks as a solution for mitigating DDoS attacks using deep packet inspection at a centralized controller. Wu *et al.* [9] studied IDS methods over massive networked systems that require real-time operation. Although these DL defenses offer high accuracy [10], their robustness is untested, and while plausible that they are vulnerable to adversarial attacks [11], the practicality of such attacks on them is unclear.

The increasing use of DL incentivizes manipulation attacks, where the DL output is an adversary's desired output. The adversary achieves this goal by applying small perturbations to the input, resulting in adversarial examples [12], [13]. The crafted adversarial examples are very similar to the original ones, and are not necessarily outside of the training data manifold, making them hard to distinguish from legitimate ones. In the context of IDSs in SDN, the failure of the anomaly detector may result in disastrous consequences, such as the failure of the entire network since a successful DDoS on the centralized controller effectively brings all the switches down.

Goal. The main goal of this study is to investigate the robustness of DL-based IDS in SDN against adversarial learning attacks. First, we incorporate generic adversarial attack methods into the anomaly detection systems. Through our analysis, we found that such adversarial learning algorithms are not applicable to the SDN, as their output adversarial flows are not realistic. Therefore, we propose Flow-Merge, an approach that is specifically designed to force DL-based IDSs in SDN to both targeted and untargeted misclassification while maintaining practical flows with malicious effects (payload). Flow-Merge combines the features of the original and a mask flow by either: accumulating or averaging.

Contributions. We make the following contributions. 1) We investigated applying generic adversarial learning methods on DL-based IDS in SDN. Our experiments demonstrate that although these methods can achieve high evasion rate, the generated adversarial flows are not realistic, precluding the applicability of generic approaches to DL-based IDSs in SDN. 2) We propose Flow-Merge, an approach specifically designed to fool DL-based IDSs in SDN, while maintaining the characteristics of original flows. Flow-Merge is able to achieve both targeted and untargeted attacks.

Organization. In Section II, we provide a brief background on SDN and adversarial learning. The proposed approach for generating practical adversarial flows is described in Section III. The performance of the proposed approach, evaluated through intensive experiments and defenses, is in Section IV. Finally, concluding remarks and future work are in Section V.

II. Preliminaries

A. Software-Defined Networking

SDN separates the control plane from data plane in network devices, which are referred to as "switches." This architecture

enables the network to implement policies from a single point, *i.e.,* the SDN controller, thus improving security, management, and decision making, as the controller has a global view of the network. SDN administrators have the capability to run applications inside the controller's platform, which in turn enables them to program switches for specific purposes [14]. The general SDN architecture and its basic traffic flow are shown in Figure 1(a) and Figure 1(b), respectively.

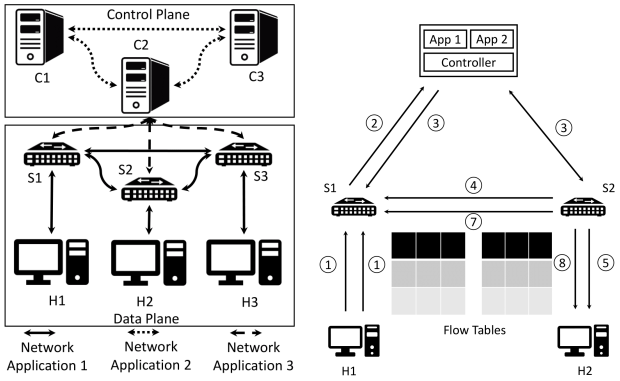
A typical SDN setup consists of a controller and one or more switches. Each switch is connected to a number of hosts and other switches. The controller communicates with the switches using a standard application program interface (southbound interface). The most common southbound interface is the OpenFlow (OF) protocol [15]. SDN applications communicate with the controller using the northbound interface (which is typically an HTTP-based interface such as REST or NETCONF), whereas a controller may communicate with another controller using the east-westbound interface, which has not been standardized yet. The controller and switches exchange various types of messages using OpenFlow. Each switch has flow tables defining flow entries of matching policy and actions to handle traffic flows within the network. Such flow tables are limited in size and populated by the controllers.

Switches take actions once an incoming packet matches a flow entry in their flow table. When a match does not exist, the packet header is directed to the controller to analyze and make a global decision, e.g., adding a new entry into the flow table. When a packet goes to the controller, this typically means the packet belongs to a flow that either was aged out or not seen before, and the controller must add an entry so that more packets from that flow do not arrive at the controller. If the packet belonged to a benign flow, the delay should be avoided, and an entry needs to be added to the flow table so that subsequent packets of the flow are properly forwarded. If the packet belongs to a malicious or disallowed flow, a flow table entry should be also added to drop subsequent packets at the switch before they get forwarded to the controller. Handling data packets at the switch and reducing the number of packets sent to the controller is key to ensuring the controller is not overloaded. Yet, merely responding to new malicious flows and adding new flow table entries is a suboptimal strategy.

Since it has a global view of the network, the controller can swiftly block attacks, specifically DDoS attacks, utilizing anomaly detection systems. On the other hand, the failure of such centralized anomaly detection systems may have disastrous consequences, e.g., the failure of an SDN controller may take down an entire network. For instance, if an adversary can forge a malicious packet that looks like a normal one, the packet will be directed to the controller and may cause the generation of a new flow entry in the flow table. Repeating that may overwhelm the southbound interface, thus affecting the entire network and not only the targeted service.

B. Adversarial Learning

Fooling a classifier is achieved by introducing a small perturbation δ into the input domain [12]. Adversarial attacks



(a) General architecture (b) Basic traffic flow

Fig. 1. A general architecture of the SDN (1(a)) and its basic traffic flow (1(b)). Here, S1:3 denotes OpenFlow Switch 1:3, C1:3 shows controller 1:3, and H1:3 represents end host 1:3.

on classifiers can be categorized from multiple perspectives, such as adversary's goals and capabilities to achieve them [12].

Adversarial Goals. The adversary's main goal can be characterized based on the inherent nature of the incorrectness into three main groups, including confidence reduction, non-targeted misclassification, and targeted misclassification.

- *Confidence reduction:* The adversary's goal is to reduce the confidence of the model such that the prediction's ambiguity increases. It should be noted that the model output does not necessarily need to be incorrect.
- *Non-targeted misclassification:* The adversary's goal is to generate Adversarial Examples (AEs) x' by applying small perturbations to the input domain x, such that the model's output $f(.)$ yields any class other than the original one ($f(x') \neq f(x)$). Note that generated AEs are similar to the original sample.
- *Targeted misclassification:* The adversary's goal is to craft AEs x' that force the model $f(.)$ to generate the adversary's desired output $f(x') = t$. Note that this type of attack is more complex than the other two.

Adversarial Capabilities. Attacks can be categorized based on the adversary's capabilities at the test time into two categories: white-box and black-box attacks. Other attacks are characterized by information accessed by the adversary:

- *Model and training data:* The adversary has full access to the training dataset and the model, including the model architecture, link weights, activation functions, etc.
- *Oracle:* The adversary has no prior knowledge about the structure of the model, but he has an oracle access to the model allowing him to conduct queries to the model to infer the relationship between input and output instances.
- *Sample:* The adversary can collect inputs/outputs pairs of an unknown model, but cannot infer the input's impact on the model's output. Note that this attack is practical only when there is a sufficient number of samples.

Threat Model. We assume an adversary with full knowledge about the baseline model, which is a convolutional neural network. Moreover, we assume the adversary's goal is to conduct

both targeted and non-targeted misclassification attacks.

III. Approach

Several DL-based approaches are proposed for IDSs in SDN [5], [6]. While it has been shown that DL models are prone to adversarial attacks, a comprehensive exploration on the impact of adversarial attacks on DL-based IDSs is lacking.

Challenges. Launching adversarial attacks on flow-based IDSs in SDN is challenging because 1) the generated adversarial flows should be realistic, can be observed by an actual packets flow, and 2) the flow-based features are highly inter-dependent. Existing generic adversarial attacks are applicable into domains where features are independent, *e.g.,* images, necessitating adversarial attack methods designed specifically for flow-based IDSs in SDN, as the perturbation generated by generic attacks is applied directly into the feature space.

Approach. We use two approaches: generic and Flow-Merge. The first approach uses generic off-the-shelf adversarial attack methods to generate AEs. We design the Flow-Merge approach to generate adversarial flows in SDN that are more realistic. For both approaches, two configurations are used: misdetection and misclassification. The first configuration does not consider the DDoS attack type, unlike the second configuration.

A. Generic Adversarial Attacks

Generic adversarial attacks were developed for image misclassification by small perturbation to the input, leading to incorrect model output. In this study, we utilized five adversarial attack algorithms: Carlini & Wanger (C&W) [16], ElasticNet [17], DeepFool [18], Momentum Iterative Method (MIM) [19], and Projected Gradient Decent (PGD) [20].

Carlini & Wagner (C&W) Method. The C&W method [16] is a gradient-based adversarial attack that optimizes the penalty and three distance metrics: L_∞, L_2, and L_0 norms. The AE generation in C&W is expressed as:

$$\min ||\delta||_p^2 \text{:}\ \ g\left(x+\delta\right)=y'\ ,\ \ x+\delta\in X,$$

where x is the input, y' is the targeted class, δ is a perturbation parameter, and $g(.)$ is the objective function. Based on C&W, the added perturbation needs to be small, and a non-targeted misclassification attack can be launched using:

$$min\,||\delta||_p^2 \text{:}\ \ g\left(x+\delta\right)\neq y\ ,\ \ x+\delta\in X.$$

Smaller L_2 corresponds to a smaller perturbation to the input, so we utilize the L_2-based C&W attack to generate AEs, and the perturbation δ is defined as $\delta = {}^1\!/_2\left(\tanh\left(w\right)+1\right)-x$, where $\tanh(.)$ and w are the hyperbolic tangent function and an auxiliary variable. The value of w is optimized using:

$$\min_w ||{}^1\!/_2\left(\tanh\left(w\right)+1\right)||_2+c\cdot g\left({}^1\!/_2\left(\tanh\left(w\right)+1\right)\right),$$

where c is a constant. The goal of C&W is to increase the generated adversarial example and the original sample similarity by minimizing the L_p norm distance between them.

DeepFool Method. DeepFool is an iterative non-targeted adversarial attack, introduced by Moosavi-Dezfooli *et al.* In DeepFool, the neural network is a linear model. In the AE generation process, the distance between the input and the corresponding class increases every iteration, and the class k with the highest probability is considered as the output of the objective function $f_k(x)$, defined as $\hat{k}(x) = \text{argmax}_k f_k(x)$.

For classifier $f(x) = W^T x + b$, the required perturbation to misclassify input x can be computed using:

$$\underset{r}{\text{argmax}}||r||_2\text{: }\exists k : w_k^T\left(x_0+r\right)+b_k \geq w_{\hat{k}(x_0)}^T\left(x_0+r\right)+b_{\hat{k}(x_0)},$$

where w_k is mapped to the k-th column of W. DeepFool can be generalized to multi-class non-linear structure using

$$\hat{l}\left(x_0\right)=\underset{k\neq\hat{k}(x_0)}{\text{argmin}}\left(|f_k\left(x_0\right)-f_{\hat{k}(x_0)}\left(x_0\right)|\right)/||w_k-w_{\hat{k}(x_0)}||_2,$$

where the output is the next closest class to x_0. Once the minimum perturbation value δ is found using

$$\delta\left(x_0\right)=\frac{|f_{\hat{l}(x_0)}\left(x_0\right)-f_{\hat{k}(x_0)}\left(x_0\right)|}{||w_{\hat{l}(x_0)}-w_{\hat{k}(x_0)}||_2^2}\left(w_{\hat{l}(x_0)}-w_{\hat{k}(x_0)}\right),$$

the adversarial example can be crafted as $x' = x + \delta$.

ElasticNet Method. Inspired by C&W, ElasticNet [17] launches L_1 distance-based non-targeted attacks. ElasticNet benefits from the same objective function as C&W, and the AE generation process is expressed as:

$$f\left(x\right)=\max\{[\text{logit}\left(x\right)]_{y_0}-\max_{j\neq y_0}[Logit\left(x\right)]_j,-k\},$$

where f is the loss function, y_0 is the original sample x_0's label, j is the current sample's label, and $\text{logit}(x) = \log(x/(1-x))$. In ElasticNet, the model's output is manipulated using two regularization factors, $\beta \geq 0$ and c, corresponding to perturbation $\delta = x - x_0$ and loss function f:

$$\min_\delta \quad c.f\left(x,y\right)+\beta||\delta||_1+||\delta||_2^2.$$

Momentum Iterative Method (MIM). MIM [19] is inspired by the Fast Gradient Sign Method (FGSM) [21], where the goal is to preserve the performance over black-box models. MIM applies momentum gradients in every iteration as:

$$\underset{x'}{\text{argmax}}\ J\left(x',y\right),\quad s.t.\ ||x'-x||_\infty\leq\epsilon$$

where J is a loss function and ϵ is a maximum distortion control. The momentum gradient (M_g) is calculated as:

$$M_{g_{t+1}}=\mu M_{g_t}+\frac{\nabla_x J_\theta\left(x_t',l\right)}{||\nabla_x J_\theta\left(x_t',l\right)||},$$

where ∇ and μ are gradient function and a decay factor, respectively. In MIM, x' is updated at each iteration using:

$$x'_{t+1}=x'_t+\epsilon\cdot sign\left(M_{g_t}+1\right).$$

Once it reaches its termination criteria, MIM returns x'_{t+1} as the AE for input x. The initial values of x'_0 and M_{g_0} are considered as the original input and 0, respectively.

Projected Gradient Descent (PGD) Method. PGD [20] benefits from an Empirical Risk Minimization (ERM) method to craft AEs while reducing the empirical risk with a trade-off of high performance using $\mathbb{E}_{(x,y)\sim D}[L\left(x,y,\theta\right)]$, where

L is the loss function and y is the label corresponding to the original input x. In PGD, the adversary is able to apply perturbation of scalar value S to the input x, and we can update the representation of the ERM to:

$$\min_{\theta} \rho(\theta) : \rho(\theta) = \mathbb{E}_{(x,y)\sim D}[\max_{\delta \in S} L(x+\delta, y, \theta)],$$

where δ denotes the added perturbation and $\rho(\theta)$ is the objective function. We note that PDG is an iterative method, where x' is updated in each iteration.

Practicality Limitations. Although the aforementioned methods excel on images, they were not designed to consider feature dependencies. For instance, manipulating a feature derived from a set of features independently may cause practicality issue as the generated feature space may not be possible to observe (due to lost dependency). Moreover, these methods focus on misclassification, regardless of the functionality, *i.e.,* a malicious adversarial flow may be misclassified as benign with a feature space representing zero packets, resulting in functionality preserving issues.

B. Flow-Merge

Approaches discussed so far for generating AEs consider features independent of one another, and attempt to alter them in a way that would result in a misclassification. In reality, however, those features are very highly dependent on one another. For example, changing the count of TCP packets as a feature would immediately alter the ratio-based features related to TCP. As such, we need a mechanism for changing the features by altering the associated flows in a consistent manner, which we achieve by Flow-Merge. Flow-Merge crafts adversarial flows that fool the DL-based DDoS defense while preserving characteristics of realistic flows. Flow-Merge modifies a flow with a representative mask flow from a selected target class. As a result of Flow-Merge, the features of the original and the mask flows are combined using one of two approaches: accumulating or averaging. The count-based features, such as the number of incoming/outgoing TCP flows, are accumulated, while the ratio-based features, such as the fraction of TCP flows over the total number of incoming/outgoing flows, are averaged in a weighted form.

The weights for each ratio-based feature are protocol-specific. For instance, the number of incoming TCP flows determine the weight of the fraction of TCP flows with the SYN flag set. At the feature level, let $X = \{x_1, \ldots, x_k\}$ be the features of the original flow, and $Y = \{y_1, \ldots, y_k\}$ be the features of the mask flow. A feature vector of the masked flow (modified one) is then calculated as $Z = \{z_1, \ldots, z_k\}$ such that $z_i = [n/(n+m)]x_i + [m/(n+m)]y_i$, where n is the number of packets associated with the studied protocol in the first flow, and m is the number of packets associated with the same protocol in the second flow. The count-based features are simply accumulated (i.e., $z_i = x_i + y_i$). Flow-Merge's design is shown in Figure 2. Z can be targeted and non-targeted; while the former requires the classifier's output to be a desired targeted class the latter only requires an incorrect output (i.e., an output other than the true class). Flow-Merge simulates the actual merge of two flows' packets, preserving the functionality and practicality of each flow.

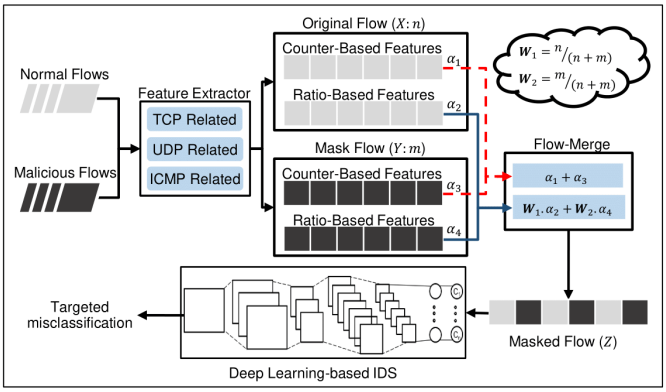


Fig. 2. A general architecture of Flow-Merge. Here, n and m are number of the TCP packets in original flow X and mask flow Y, respectively. In this figure, $W_1 = (n/(n+m))$ and $W_2 = (m/(n+m))$, while α_2 and α_4 are the ratio-based features of X and Y, respectively.

IV. Evaluation and Discussion

A. Dataset

For benchmarking, we use the dataset in [5] to evaluate the performance of the methods in this study. To collect the dataset, traffic traces of various online activities, including web browsing, audio/video streaming, real-time messengers, and online gaming were collected using tcpdump [22] on a Linux system and using port mirroring on a Wi-Fi access point for 72 hours. The first 48 hours were used as normal flows and the rest were mixed with attack traffic. To collect the attack traffic, a private network consisting of 10 DDoS attackers and 5 victim hosts in a segregated laboratory environment using VMWare ESXi host has been created. The different kinds of DDoS attacks with different packet frequencies and sizes were launched using hping3 [23]. An SDN testbed on the same ESXi host was created. The testbed consists of an SDN controller, an OF switch, and a network host using the Ubuntu Linux systems. POX [24] is used as the controller, while OpenvSwitch [25] is used as the switch. The traffic traces for normal and attack traffic were replayed, mixing the last 24 hours of normal flow with the attack traffic, one at a time, in the host system using tcpreplay [26]. The data is saved using intervals of 60 seconds and divided into training and test sets. The distribution of the dataset across different classes is shown in Table IV and more information on the data and its collection can be found in [5]. In total, a set of 68 statistical features, divided into TCP related features (Table I), UDP related features (Table II), and ICMP related features (Table III), were extracted for each record.

B. Experimental Setup

Evaluation system. All experiments were conducted using Python 3.6 running on a Ubuntu 16.04 system with i5-8500 CPU (3.00GHz), 32GB DDR4 RAM, 512GB SSD storage, and NVIDIA GTX980 Ti GPU for DL processing.

TABLE I
FEATURES EXTRACTED FROM TCP FLOWS. HERE, T REFERS TO TYPE OF THE FEATURES: C REFERS TO COUNT-BASED FEATURES, R REFERS TO RATIO-BASED FEATURES, (I) FOR INCOMING, AND (O) FOR OUTGOING.

#	T	Feature Description	#	T	Feature Description
1	C	# TCP flows (i)	18	R	src port entropy (i)
2	R	TCP flows over total (i)	19	C	# distinct dst ports (i)
3	C	# TCP flows (o)	20	R	dst ports entropy (i)
4	R	TCP flows over total (o)	21	R	dst ports $\leq$ 1024 (i)
5	R	symmetric flows (i)	22	R	dst port $>$ 1024 (i)
6	R	Asymmetric flows (i)	23	R	Flows (i), SYN set
7	C	# distinct src IP (i)	24	R	Flows (o), SYN set
8	R	src IP entropy (i)	25	R	Flows (i), ACK set
9	R	Bytes per flows (i)	26	R	Flows (o), ACK set
10	R	Bytes per flows (O)	27	R	Flows (i), URG set
11	R	# packets per flows (i)	28	R	Flows (o), URG set
12	R	# packets per flows (o)	29	R	Flows (i), FIN set
13	C	# distinct window size (i)	30	R	Flows (o), FIN set
14	R	Entropy of window size (i)	31	R	Flows (i), RST set
15	C	# distinct TTL values (i)	32	R	Flows (o), RST set
16	R	Entropy of TTL values (i)	33	R	Flows (i), PUSH set
17	C	# distinct src ports (i)	34	R	Flows (o), PUSH set

TABLE II
FEATURES EXTRACTED FROM UDP FLOWS. HERE, T REFERS TO TYPE OF THE FEATURES: C REFERS TO COUNT-BASED FEATURES, R REFERS TO RATIO-BASED FEATURES, (I) FOR INCOMING, AND (O) FOR OUTGOING.

#	T	Feature Description	#	T	Feature Description
35	C	# UDP flows (i)	45	R	# packets per flows (i)
36	R	UDP flows over total (i)	46	R	# packets per flows (o)
37	C	# UDP flows (o)	47	C	# distinct src ports (i)
38	R	UDP flows over total (o)	48	R	src ports entropy (i)
39	R	Symmetric UDP flows (i)	49	C	# of distinct dst ports (i)
40	R	Asymmetric UDP flows (i)	50	R	dst ports entropy (i)
41	C	# distinct src IP (i)	51	R	dst ports $\leq$ 1024 (i)
42	R	Entropy of src IP (i)	52	R	dst port $>$ 1024 (i)
43	R	Bytes per flows (i)	53	C	# distinct TTL values (i)
44	R	Bytes per flows (i)	54	R	TTL values entropy (i)

TABLE III
FEATURES EXTRACTED FROM ICMP FLOWS. HERE, T REFERS TO TYPE OF THE FEATURES: C REFERS TO COUNT-BASED FEATURES, R REFERS TO RATIO-BASED FEATURES, (I) FOR INCOMING, AND (O) FOR OUTGOING.

#	T	Feature Description	#	T	Feature Description
55	C	# ICMP flows (i)	62	R	src IP entropy (i)
56	R	ICMP flows over total (i)	63	R	Bytes per flows (i)
57	C	# ICMP flows (o)	64	R	Bytes per flows (o)
58	R	ICMP flows over total (o)	65	R	# packets per flows (i)
59	R	Symmetric ICMP flows (i)	66	R	# packets per flows (o)
60	R	Asymmetric ICMP flows (i)	67	C	# distinct TTL values (i)
61	C	# of distinct src IP (i)	68	R	TTL values entropy (i)

TABLE IV
PER-CLASS FLOW RECORDS DISTRIBUTION.

Class types		# of records	
		Train	Test
Normal		49,179	21,076
DDoS attacks	TCP	5,471	2,344
	UDP	5,273	2,260
	ICMP	1,602	686
	TCP & UDP	4,694	2,011
	TCP & ICMP	4,739	2,031
	UDP & ICMP	4,437	1,902
	All (TCP & UDP & ICMP)	5,615	2,407
Total		81,010	34,717

Intrusion Detection System. We trained two models for detection and classification. The detection model is a two-class classifier, while the classification model consists of eight classes: one normal class, and seven DDoS attack types. The models are based on CNNs, each of which consists of multiple layers, including convolutional, activation, pooling, and dropout. Our tested IDS consists of three main blocks: convolutional block 1 (CB1), convolutional block 2 (CB2), and classification block (CL). CB1 and CB2 are responsible for deep feature extraction, while CL does the classification. More detailed description of these blocks is in the following.

CB1: The input (X) convolves with 68 filters ${F_1}'$ of size 1×3 in C_1 layer with padding, resulting in a 2D tensor of size 68×68. The tensor is fed into C_2, which is 1D convolutional layer without padding and 68 filters of size 1×3, resulting in a 2D tensor ${C_2}''$ of size 66×68. A max pooling with size and stride of 2 and dropout with probability of 0.25 are then applied, resulting in a 2D tensor S_1 of size 33×68.

CB2: This block is similar to CB1 except that the number of filters in the convolutional layers is doubled. The output of CB1 S_{b1} is fed into a 1D convolutional layer with padding and 136 filters of size 1×3, convolving over the data with a stride of 1, resulting in a 2D tensor ${C_3}''$ of size 33×136. The tensor is fed into C_4, which is a 1D convolutional layer without padding and 136 filters of size 1×3, resulting in a 2D tensor ${C_4}''$ of size 31×136. A max pooling with size and stride of 2 and dropout with probability of 0.25 are applied, resulting in a 2D tensor S_2 of size 15×136.

CL: The output of CB2 is fed into a classification block, flattened and fed into a dense layer of size 512, resulting into a fully connected feature map layer FCL. FCL is fed into a dropout with a probability of 0.5 followed by a softmax classifier. The output of the softmax is evaluated based on standard evaluation metrics, e.g., accuracy rate (AR), false negative rate (FNR), and false positive rate (FPR).

We used Rectified Linear Units (ReLU) activation function for both the convolutional and fully-connected layers. To prevent model over-fitting, we used dropout. The general design of our baseline DL-based IDS is depicted in Figure 3. More information regarding CNNs can be found in [27].

Adversarial Attacks Configuration. The goal of the attacks is to fool the DL-based IDS by crafted AEs, while maintaining their practicality: (1) preserving the functionality of the flow and (2) can be extracted from a designed flow. The following are configurations of the attacks we used in our evaluation. All of the attacks are implemented using Cleverhans [28].

C&W: We use the L_2 distance iterative method by setting the number of iterations and learning rate to 200 and 0.1.

ElasticNet: AEs are generated based on L_1 distance and using the same loss function of the C&W. To generate AEs, we set the iterations number to 250 with 0.1 learning rate.

DeepFool: We set the number of iterations and overshooting value to 250 and 0.02.

MIM: We set the number of iterations to 250 with ϵ of 0.3.

PGD: To generate AEs using PGD, we set the number of iterations and distortion (ϵ) rate to 250 and 0.3, respectively.

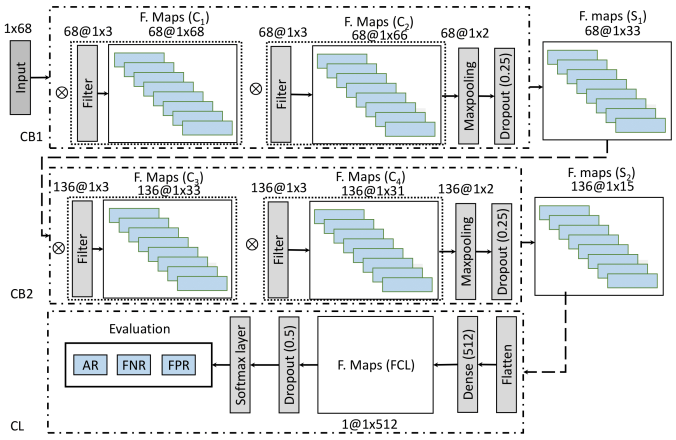


Fig. 3. The internal design of our DL-based IDS, which consists of multiple convolutional layers followed by a softmax classifier. Here, number of filters (A) and their size (BxC) are shown as A@BxC.

TABLE V

MISCLASSIFICATION RATE OF EACH MODEL AGAINST ADVERSARIAL ATTACKS. ACC REFERS TO THE ACCURACY OF THE BASELINE MODEL, EN REFERS TO ELASTICNET METHOD, D. CONF. REFERS TO THE DETECTION MODEL., AND C. CONF. REFERS TO THE CLASSIFICATION MODEL.

Model	Baseline Model(%)			Attack Misclassification Rate (%)				
	Acc	FNR	FPR	C&W	EN	DF	MIM	PGD
D. Conf.	99.83	0.05	0.34	98.92	99.84	62.78	93.65	99.76
C. Conf.	96.05	8.18	0.54	95.24	97.13	47.14	88.57	99.26

TABLE VI

MISCLASSIFICATION RATE OF THE MODEL OVER EACH ATTACK CLASS. HERE, `All`* CONTAINS ALL TCP, UDP, AND ICMP FEATURES.

Method	Attack Misclassification Rate (%)						
	TCP	UDP	ICMP	TCP/UDP	TCP/ICMP	UDP/ICMP	`All`*
C&W	98.76	99.29	99.27	82.39	84.24	96.84	88.57
ElasticNet	98.40	99.24	99.27	83.40	84.83	97.10	90.19
DeepFool	86.05	90.48	70.26	72.55	79.03	78.97	82.09
MIM	99.83	**100**	**100**	**99.40**	**97.30**	99.94	**99.58**
PGD	**99.70**	**100**	**100**	96.42	96.65	**100**	95.93

C. Results and Discussion

We present our results: the DL-based IDS (baseline), the generic adversarial attacks, and our Flow-Merge attack.

DL-based IDS. We designed two configurations: the anomaly detection and the anomaly classification. The detection model distinguishes malicious flows from normal flows, regardless of the attack type, while the misclassification model classifies flows as normal traffic or one of the seven types of network attacks (TCP, UDP, ICMP, or their combinations). The models are trained over 68 statistical features extracted from traffic flows directed to the SDN controller. In the detection model, we achieved an accuracy rate of 99.83% with FNR of 0.05% and FPR of 0.34%. Our classification model achieved an accuracy rate of 96.05% with FNR of 8.18% and FPR of 0.54%. Note that the classification accuracy is slightly less than the detection accuracy, because different attack types share similar patterns, making them difficult to differentiate.

Generic Adversarial Attack. We implemented five generic adversarial learning methods. Using those methods, we were able to generate AEs that erroneously classify malicious flows as benign flows and vice versa. Our evaluations demonstrate that we can achieve a misdetection rate as high as 99.84% using the ElasticNet method and as low as 62.78% using the DeepFool adversarial learning method. Moreover, the achieved misclassification rate varies between 99.26% and 47.14%, using both PGD and DeepFool, respectively.

Except with DeepFool, all other attack methods achieve high misclassification rates, showing the vulnerability of DL-based IDSs in SDN. The lower misclassification rate of DeepFool is explained by its attempt to minimize the distance between the crafted AE and the original samples through minimizing the perturbation size. The higher performance of other algorithms, such as MIM and PGD, is due to their intrinsic characteristics. MIM, for example, is an iterative method that adds small perturbations in each iteration until it reaches misclassification or the maximum iterations. PGD, on the other hand, tries to craft AEs under minimized empirical risk with a trade-off of high performance cost. The detailed distribution of the misclassification rate of the classification approach over each class for each attack method is shown in Table VI.

Flow-Merge. Given the main objective of Flow-Merge, we focus on a targeted misclassification attack by masking classes using a target class. The masks are selected based on the dominance of the flows as either TCP, UDP, or ICMP protocols. The experiments are carried out for both detection and classification. For detection, the goal is to misclassify the malicious flow into benign and vice versa, this is done by merging the features of the original flow with the selected flow from the targeted class using Flow-Merge. Table VII shows the detailed results of misdetection rates using Flow-Merge for each dominant flow. The results show that while the adversary can easily forge a malicious flow classified as a benign flow for all three dominant flows, the adversary can only forge a TCP dominant benign flow classified as malicious. The reason for this outcome is that the number of TCP dominant flows (65.81%) is far more than the number of UDP (20.45%) and ICMP (13.72%) dominant flows. It should be noted that the larger number of TCP dominant flows may result in better learning of the DL-based IDSs, while perturbation of such flows increases the chance of the adversary to achieve higher misclassification rate, as our findings clearly demonstrate.

For misclassification, we observed that it is difficult to force the model to yield specific outputs. For example, forcing the model to label a TCP attack record as a UDP attack record is almost impossible (misclassification rate of 0%). The reason behind this performance is the nature of the TCP attack, which is different from the UDP attack, in aspects concerning, among others, packet header flags, demonstrated by the associated group of features. We also observed a misclassification rate of 100% in other cases, *i.e.,* misclassifying all attack classes to benign or `All`* attack. The reason behind this behavior is that both classes contain all protocols (TCP, UDP, and ICMP) within their flows, which leads to a higher success rate of misclassifying a single or multiple protocol attacks to these

TABLE VII
MISDETECTION USING FLOW-MERGE FOR EACH DOMINANT FLOW. COLUMNS ARE ORIGINAL LABELS, AND ROWS ARE PREDICTED CLASSES.

Flow type		Benign	Malicious
TCP	Benign	0.003	0.986
	Malicious	0.997	0.014
UDP	Benign	0.888	0.818
	Malicious	0.112	0.182
ICMP	Benign	0.892	0.939
	Malicious	0.108	0.061

TABLE VIII
MISCLASSIFICATION TO BENIGN USING TCP, UDP, AND ICMP DOMINANT FLOWS. COLUMNS REFER TO THE ORIGINAL LABEL AND ROWS REFER TO THE PREDICTED CLASSES. C0 REPRESENTS A NORMAL FLOW, WHILE C1-C7 REPRESENT THE DDOS ATTACKS IN TABLE IV.

Flow type		C0	C1	C2	C3	C4	C5	C6	C7
TCP	C0	1	1	1	1	1	1	1	1
	C1–C7	0	0	0	0	0	0	0	0
UDP	C0	1	1	1	1	0.989	0.996	1	0.999
	C1	0	0	0	0	0	0	0	0
	C2	0	0	0	0	0.004	0	0	0.001
	C3	0	0	0	0	0.004	0	0	0
	C4–C5	0	0	0	0	0	0	0	0
	C6	0	0	0	0	0.001	0.004	0	0
	C7	0	0	0	0	0.002	0	0	0
ICMP	C0	1	1	1	1	0.999	1	1	1
	C1–C6	0	0	0	0	0	0	0	0
	C7	0	0	0	0	0.001	0	0	0

classes. The detailed results of misclassification rates using Flow-Merge are shown in Table VIII, Table IX, and Table X, where the columns and rows represent the actual and predicted labels, respectively. In these tables, C0 represents a normal flow, while C1-C7 represent the DDoS attacks in Table IV.

D. Defense via Adversarial Training

To investigate how the generated AEs through Flow-Merge are able to evade detection, we implemented a defense configuration. The main goal of the defense configuration is to improve the robustness of the IDSs against adversarial examples, even those crafted by Flow-Merge. There are several defensive methods in the literature, including defensive distillation [29] and adversarial training [21], [30], [31]. In this work, we used adversarial training, which is one of the most successful robust methods by far [32], to investigate the robustness of our IDS against attacks. The idea of adversarial training is to inject AEs into the model's training phase to increase its robustness against those AEs [32]. The goal of the adversarial training is to solve the following adversarial empirical risk minimization:

$$\min_{\theta} \; \mathrm{E}_{(x,y)\sim\widetilde{D}} \left[max_{\delta \epsilon S} \; L(x+\delta, y; \theta) \right],$$

where x is the input, y is the model output, θ is the model parameters, and δ is a small perturbation. In our analysis, we considered two different assumptions: 1) where the defender knows the characteristics of the mask (a more realistic assumption), and 2) where the defender does not know anything about the mask. We conducted our analysis considering both single class and all classes in adversarial training.

TABLE IX
MISCLASSIFICATION TO *All* ATTACK USING A TCP, UDP, AND ICMP DOMINANT FLOWS (ABBREVIATIONS ARE SIMILAR TO TABLE VIII).

Flow type		C0	C1	C2	C3	C4	C5	C6	C7
TCP	C0	0.007	0	0	0	0	0	0	0
	C1–C4	0	0	0	0	0	0	0	0
	C5	0.002	0	0	0	0	0	0	0
	C6	0	0	0	0	0	0	0	0
	C7	0.99	1	1	1	1	1	1	1
UDP	C0	0.006	0	0	0	0.001	0.001	0	0
	C1–C6	0	0	0	0	0	0	0	0
	C7	0.993	1	1	1	0.999	0.999	1	1
ICMP	C0	0.023	0	0	0	0	0	0	0
	C1–C3	0	0	0	0	0	0	0	0
	C4	0	0	0	0	0	0.001	0	0
	C5–C6	0	0	0	0	0	0	0	0
	C7	0.977	1	1	1	0	0.999	1	1

TABLE X
MISCLASSIFICATION TO TCP ATTACK USING A TCP, UDP, AND ICMP DOMINANT FLOWS (ABBREVIATIONS ARE SIMILAR TO TABLE VIII).

Flow type		C0	C1	C2	C3	C4	C5	C6	C7
TCP	C0	0.207	0	0.648	0.209	0.292	0.261	0.841	0.645
	C1	0.793	1	0.267	0.041	0.581	0.321	0	0.152
	C2	0	0	0	0	0	0	0	0
	C3	0	0	0	0	0	0.001	0	0
	C4	0	0	0	0.372	0.032	0.116	0.005	0.046
	C5	0	0	0.006	0.119	0.003	0.177	0.002	0.016
	C6	0	0	0	0.026	0	0.033	0	0.002
	C7	0	0	0.079	0.232	0.091	0.091	0.152	0.139
UDP	C0	0.993	0.033	1	1	0.365	0.356	0.999	0.402
	C1	0.007	0.967	0	0	0.024	0.008	0	0
	C2	0	0	0	0	0.001	0	0	0
	C3	0	0	0	0	0.003	0.001	0	0
	C4	0	0	0	0	0.523	0.014	0	0
	C5	0	0	0	0	0.051	0.427	0	0.013
	C6	0	0	0	0	0	0	0	0
	C7	0	0	0	0	0.034	0.194	0.001	0.585
ICMP	C0	0.993	0.033	1	1	0.365	0.356	0.999	0.402
	C1	0.007	0.967	0	0	0.024	0.008	0	0
	C2	0	0	0	0	0.001	0	0	0
	C3	0	0	0	0	0.003	0.001	0	0
	C4	0	0	0	0	0.523	0.014	0	0
	C5	0	0	0	0	0.051	0.427	0	0.013
	C6	0	0	0	0	0	0	0	0
	C7	0	0	0	0	0.034	0.194	0.001	0.585

Our results are shown in Table XI, where it is evident that the adversarial training approach is able to improve the robustness of the model only for those types of AEs considered in the adversarial training. For example, when retraining the model with AEs of class C0, the model is able to detect 89.90% of the AEs of the same class. The model, however, fails to detect AEs of other classes (15.05%). Moreover, changing the mask would decrease the robustness of the model, *i.e.,* training on AEs generated by a specific mask, and classifying AEs generated by another mask from the same class. The overall performance of the adversarial training approach once all types of AEs (C0 to C7) are considered in the adversarial training are not promising (63.01% and 49.31% for known and unknown masks, respectively), highlighting the effectiveness of the generated AEs of our proposed Flow-Merge approach, and leaving customized defenses as a potential future work.

V. CONCLUSION

We investigated the robustness of DL-based DDoS defenses in SDN against adversarial attacks, whereby an adversary

TABLE XI
RESULTS OF ADVERSARIAL TRAINING APPROACH. THE FLOW-MERGE IS ABLE TO REMAIN UNDETECTED OVER ADVERSARIAL TRAINING APPROACH.

Mask	Crafted AEs	C0	C1	C2	C3	C4	C5	C6	C7	C0-C7
Known	Same class	89.90	81.58	61.83	75.89	87.39	93.09	85.77	98.01	63.01
	Other classes	15.05	40.43	44.88	45.18	37.07	34.17	42.09	34.24	
Unknown	Same class	67.57	81.60	39.38	47.60	85.48	89.30	52.44	95.07	49.31
	Other classes	19.38	21.95	27.97	29.20	22.13	20.25	28.11	18.89	

attempts to force the DL model into misclassification by introducing small perturbations. Because generic AE generation algorithms are shown to produce unrealistic flows, Flow-Merge was proposed. Flow-Merge utilizes a weighted merging technique over ratio-based features to craft adversarial inputs. The evaluation results show a high misclassification rate of 99.84% using generic adversarial attacks for untargeted misclassification. Moreover, Flow-Merge produces realistic adversarial flows for targeted misclassification with a success rate of 100%. Adversarial training, a widely used defense mechanism against AEs, is shown effective on generic attacks, although limited against Flow-Merge. In the future, we will explore effective defense mechanisms against Flow-Merge.

Acknowledgement. This work is supported in part by NRF-2016K1A1A2912757 and NVIDIA GPU Grant, and NSF awards 1647189, 1814086, and 1643207.

REFERENCES

[1] L. Schehlmann, S. Abt, and H. Baier, "Blessing or curse? revisiting security aspects of software-defined networking," in *Proceedings of the Network and Service Management*, 2014, pp. 382–387.

[2] S. Shin, L. Xu, S. Hong, and G. Gu, "Enhancing network security through software defined networking (sdn)," in *International Conference on Computer Communication and Networks*, 2016, pp. 1–9.

[3] K. Kalkan, G. Gur, and F. Alagoz, "Defense mechanisms against ddos attacks in sdn environment," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 175–179, September 2017.

[4] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 602–622, Firstquarter 2016.

[5] Q. Niyaz, W. Sun, and A. Y. Javaid, "A deep learning based ddos detection system in software-defined networking (SDN)," *ICST Transactions on Security and Safety*, 2017.

[6] A. Abubakar and B. Pranggono, "Machine learning based intrusion detection system for software defined networks," in *Proceedings of the Seventh International Conference on Emerging Security Technologies (EST)*, 2017, pp. 138–143.

[7] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep recurrent neural network for intrusion detection in sdn-based networks," in *Proceedings of the 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 202–206.

[8] M. E. Ahmed, H. Kim, and M. Park, "Mitigating dns query-based ddos attacks with machine learning on software-defined networking," in *Proceedings of Military Communications Conference*, 2017.

[9] K. Wu, Z. Chen, and W. Li, "A novel intrusion detection model for a massive network using convolutional neural networks," *IEEE Access*, 2018.

[10] H. Alasmary, A. Khormali, A. Anwar, J. Park, J. Choi, A. Abusnaina, A. Awad, D. Nyang, and A. Mohaisen, "Analyzing and Detecting Emerging Internet of Things Malware: A Graph-based Approach," *IEEE Internet of Things Journal*, 2019.

[11] A. Abusnaina, A. Khormali, H. Alasmary, J. Park, A. Anwar, and A. Mohaisen, "Adversarial learning attacks on graph-based iot malware detection systems," in *Proceedings of the 39th IEEE International Conference on Distributed Computing Systems, ICDCS*, 2019.

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

[13] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2574–2582.

[14] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

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

[17] P. Chen, Y. Sharma, H. Zhang, J. Yi, and C. Hsieh, "EAD: elastic-net attacks to deep neural networks via adversarial examples," in *Proceedings of Conference on Artificial Intelligence*, 2018, pp. 10–17.

[18] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2574–2582.

[19] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9185–9193.

[20] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proceedings of the 2018 International Conference on Learning Representations.*, 2018.

[21] C. S. Ian J. Goodfellow, Jonathon Shlens, "Explaining and harnessing adversarial examples," in *Proceedings of International Conference on Learning Representations.*, 2015.

[22] Tcpdump, "Tcpdump/libpcap public repository," Sep 2010. [Online]. Available: http://www.tcpdump.org/

[23] "Home – hping network security tool." [Online]. Available: http://wiki.hping.org/

[24] [Online]. Available: https://openflow.stanford.edu/display/ONL/POXWiki

[25] "Production quality, multilayer open virtual switch." [Online]. Available: http://www.openvswitch.org/

[26] [Online]. Available: http://tcpreplay.appneta.com/

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of Advances in neural information processing systems*, 2012, pp. 1097–1105.

[28] N. Papernot, N. Carlini, I. Goodfellow, R. Feinman, F. Faghri, A. Matyasko, K. Hambardzumyan, Y.-L. Juang, A. Kurakin, R. Sheatsley *et al.*, "cleverhans v2. 0.0: an adversarial machine learning library," *arXiv preprint arXiv:1610.00768*, 2016.

[29] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proceedings of IEEE Symposium on Security and Privacy, SP*, 2016, pp. 582–597.

[30] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proceedings of International Conference on Learning Representations.*, 2014.

[31] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 506–519.

[32] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "There is no free lunch in adversarial robustness (but there are unexpected benefits)," *arXiv preprint arXiv:1805.12152*, 2018.