IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 31, 2018, accepted June 27, 2018, date of publication July 9, 2018, date of current version July 30, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2854599

Deep Learning-Based Intrusion Detection With Adversaries

ZHENG WANG
National Institute of Standards and Technology, Gaithersburg, MD 20899, USA
(e-mail: zhengwang98@gmail.com)

ABSTRACT Deep neural networks have demonstrated their effectiveness in most machine learning tasks, with intrusion detection included. Unfortunately, recent research found that deep neural networks are vulnerable to adversarial examples in the image classification domain, i.e., they leave some opportunities for an attacker to fool the networks into misclassification by introducing imperceptible changes to the original pixels in an image. The vulnerability raises some concerns in applying deep neural networks in security-critical areas, such as intrusion detection. In this paper, we investigate the performances of the state-of-the-art attack algorithms against deep learning-based intrusion detection on the NSL-KDD data set. The vulnerabilities of neural networks employed by the intrusion detection systems are experimentally validated. The roles of individual features in generating adversarial examples are explored. Based on our findings, the feasibility and applicability of the attack methodologies are discussed.

INDEX TERMS Intrusion detection, neural networks, classification algorithms, data security.

I. INTRODUCTION

Today's Internet is growingly endangered by various cyber threats. Hackers are inventing new techniques on a daily basis to bypass security layers and avoid detection. Intrusion Detection Systems (IDS) are playing an indispensable role in defending against intrusions and malicious activities. IDS can be considered as two main categories based on operational logic: (1) signature-based IDS where the traffic is compared against a database of signatures of known threats; (2) anomaly-based IDS which inspects the traffic based on the behavior of activities.

Machine Learning (ML) techniques are changing our view of the world and they are impacting all aspects of our daily life. It is envisioned to change the landscape of information security, with intrusion detection included. In the past decade, a number of machine learning techniques have been applied to intrusion detection with the hope of improving detection rates and adaptability. However, most systems built based on such techniques suffer from the dependency on domain knowledge, insufficient learning capability with big data, and lack of modularity and transferability. To address those challenges in intrusion detection, deep neural networks (deep learning) recently found their success.

Deep learning requires less hand engineered features and expert knowledge. Driven by the emergence of big data and hardware acceleration, the intricacy of data can be extracted with higher and more abstract level representation from raw input features. Some recent work showed that deep learning based IDS has striking learning capability or outperforms traditional ML-based counterparts.

However, deep learning in an adversarial environment requires us to anticipate that an adversarial opponent will try to cause deep learning to fail in many ways. In many cases, the adversary is able to poison the learner's classifications, often in a highly targeted manner. For instance, an adversary can craft input data with imperceptible deviations in order to cause the learner to learn an incorrect decision-making function such as avoiding detection of attacks or causing benign input to be classified as attack input.

Recent studies confirmed that deep learning is vulnerable against well-manipulated adversarial samples in image-based datasets. While there was some pioneering work on the application of deep learning to intrusion detection, we still hardly know the vulnerability of deep learning in the intrusion detection domain against adversarial examples. In this paper, we will present a comprehensive study of the deep learning based intrusion detection with adversaries. We will evaluate the state-of-the-art attack algorithms against deep learning based intrusion detection on the NSL-KDD dataset. Based on the implementation of deep neural networks using TensorFlow, we will validate the vulnerabilities of neural networks under attacks on IDS. To gain insights into the nature of intrusion detection and its attacks, we will also explore the roles of individual features in generating adversarial examples.

II. RELATED WORK

It is demonstrated that the deep learning based approaches are helpful to overcome the challenges of developing an effective IDS such as the difficulty of feature selection and representation, and the limited availability of labeled traffic dataset. For example, Shone *et al.* [1] proposed a new deep learning classification model, Yin *et al.* [2] proposed to use recurrent neural networks for intrusion detection tasks, Javaid *et al.* [3] developed a deep learning based technique for self-taught learning in the IDS classification, and Tang *et al.* [4] applied a deep learning approach for flow-based anomaly detection. However, little attention in previous work was given to the risks posed by the emerging adversarial deep learning against IDS. The vulnerability discovered in recent years greatly limit the application of deep neural networks in security-critical areas such as self-driving, safety-critical voice-controllable systems, and IDS.

Szegedy *et al.* [6] first revealed in 2014 the intriguing discovery that deep neural networks are vulnerable to adversarial examples. They also successfully generated adversarial examples through the use of the box-constrained Limited memory approximation of Broyden-Fletcher-Goldfarb-Shanno (LBFGS) optimization algorithm. Since then, the profound implications of the findings triggered a wide interest of academia and industry in developing adversarial attacks and studying their defenses. To ease the high computational cost of the LBFGS approach, Goodfellow *et al.* [7] proposed the Fast Gradient Sign Method (FGSM). FGSM generates adversarial perturbations based on the gradient of the loss function relative to the input image and thus enables computational efficiency through backpropagation. Kurakin *et al.* [8] extended the fast gradient sign method by running a finer optimization (smaller change) for multiple iterations. Papernot *et al.* [9] created adversarial saliency maps by computing forward derivatives which are used to identify the input feature to be perturbed towards the target class. Moosavi-Dezfooli *et al.* [10] proposed an approach to find the closest distance from original input to the decision boundary of adversarial examples. Carlini and Wagner [11] introduced three new gradient-based attack algorithms (L_2, L_∞, and L_0) that are more effective than all previously known methods in terms of the adversarial success rates achieved with minimal perturbation amounts. Their L_2 attack uses a logits-based objective function which is different from all existing L_2 attacks, and avoids the box constraint by changing variables. Their L_∞ and L_0 are based on the L_2 attack and tailored to different distance metrics. The existing literature mainly deals with the art of fooling the deep neural networks for the typical computer vision tasks, e.g. recognition, and their effectiveness is demonstrated using standard image datasets, e.g. MNIST [5].

III. BACKGROUND

A. NEURAL NETWORKS AND NOTATION

Deep neural network is a machine learning algorithm powered by many layers (“deep”) of connected networks. It is often called end-to-end machine learning where sophisticated patterns are extracted from the representation of multiple simple features with limited prior knowledge. Therefore deep learning models are increasingly used to solve complicated big data problems which are often not well addressed by conventional machine learning algorithms.

A deep neural network is formed as a function $f(\cdot), f \in F : \mathbb{R}^n \to \mathbb{R}^m$. The parameters of deep learning model f is θ which is often subject to training for the objective results. The most common training process of deep learning model f aims at minimizing a loss function (e.g., cross-entropy) J.

As supervised machine learning, deep learning has two most common tasks: classification or regression. In classification problems, a discrete number of values is predicted. In regression problems, continuous valued output is predicted. In the paper, deep neural networks are used as a m-class classifier with its outputs as label of class in the classification problem, $l = 1, 2, \ldots, m$. The output of the network is computed using the softmax function, which ensures that the output vector y satisfies $0 \leq y_i \leq 1$ and $y_1 + \ldots + y_m = 1$. The output vector y is thus treated as a probability distribution, i.e., $y_i = f(x)_i$ is treated as the probability that input x has class i. The classifier assigns the label $C(x) = \text{argmax}_i f(x)_i$ to the input x. Let $C^*(x)$ be the correct label of x. The inputs to the softmax function $f(x)$ are called *logits*.

We define F to be the full neural network including the softmax function, $Z(x) = z$ to be the output of all layers except the softmax (so z are the *logits*), and

$$F(x) = softmax(Z(x)) = y \tag{1}$$

A neural network layer consists of a set of perceptrons. Each perceptron transforms a set of inputs with linear weights (and biases) and then a non-linear activation function. The multiple layers of a deep neural network are chained:

$$F = softmax \circ F_n \circ F_{n-1} \circ \ldots \circ F_1 \tag{2}$$

Where

$$F_i(x) = \sigma(\tilde{\theta}_i \cdot x) + \hat{\theta}_i \tag{3}$$

For some non-linear activation function σ, some matrix $\tilde{\theta}_i$ of model weights, and some vector $\hat{\theta}_i$ of model biases. As the model parameters, $\theta = \{\tilde{\theta}_i, \hat{\theta}_i\}$ are tunable and trainable in the machine learning process. Common choices of σ are ReLU, tanh, and sigmoid. In this paper we focus primarily on networks that use a ReLU activation function, as it currently is the most widely used activation function.

B. ADVERSARIAL EXAMPLES

We categorize the methodologies for generating adversarial examples in two dimensions in this section.

Target of adversarial examples:

- *Targeted attacks*. Considering an original input x and its target class $l = C^*(x)$, the objective of adversaries is to find a perturbed input x' satisfying $C(x') = l$ yet x, x' are very similar to each other according to some

distance metric. Such an example x' is defined as a targeted adversarial example (for a target class l).

- *Untargeted attacks*. Instead of classifying x as a given target class, we only search for an input x' so that $C(x') \neq C^*(x)$ and x, x' are close. Non-targeted attacks are strictly less powerful than targeted attacks.

Knowledge about the target model:

- *White-box attacks*. It assumes the adversary knows everything related to the trained neural network model: training data, network architectures, hyper-parameters, numbers of layers, functions of activations, network weights, etc. Many adversarial examples are generated by calculating network gradients. Since deep neural networks tend to require only raw input data without handcrafted features and to deploy end-to-end structure, feature selection is not necessary compared to adversarial examples in machine learning.
- *Black-box attacks*. It assumes the adversary has no access to the trained neural network model. The adversary, acting as a standard user, only knows the output of the model (label or confidence score). This assumption is common for attacking online Machine Learning services. Most adversarial example attacks are white-box attacks. However, they can be transferred to attack black-box services due to the transferability of adversarial examples.

C. DISTANCE METRICS

In our definition of adversarial examples, similarity between an adversarial example and its original counterpart is measured using a distance metric. There are three widely-used distance metrics in the literature for generating adversarial examples, all of which are L_p norms.

The L_p distance is written as $\| x - x' \|_p$, where the p-norm $\| \cdot \|_p$ is defined as

$$\| v \|_p = \left(\sum_{i=1}^{n} |v_i|^p\right)^{\frac{1}{p}} \tag{4}$$

Specifically,

- L_0 distance measures the number of features i such that $x_i \neq x'_i$. Therefore the L_0 distance means the number of features that have been perturbed between the two samples.
- L_2 distance measures the standard Euclidean (root mean square) distance between x and x'. A large L_0 deviation may maintain a small L_2 when slight changes are introduced to many samples.
- L_∞ distance measures the maximum change to any of the features:

$$\| x - x' \|_\infty = \max(|x_1 - x'_1|, \ldots, |x_n - x'_n|) \tag{5}$$

A large L_∞ does not necessarily cause a large L_2 if, e.g., the L_∞ is exclusively caused by a dramatic change to a single feature out of a large set of features which have minor enough perturbations. Similarly, a large L_∞ may co-exist with a small L_0 considering the case that a single feature is changed significantly while the other features remain unchanged.

IV. ATTACK ALGORITHMS

A. FAST GRADIENT SIGN METHOD (FGSM)

While there was an early proposal to use linear search method to find adversarial examples [6], the linear search method was often too computation intensive to be affordable in practice. To ease the search for adversarial examples, Goodfellow et al. proposed a fast method for generating adversarial examples called Fast Gradient Sign Method (FGSM) [7]. They only performed one step gradient update along the direction of the sign of gradient at each pixel. Their perturbation can be expressed as:

$$\eta = \epsilon sign(\nabla_x J_\theta(x, l)) \tag{6}$$

where ϵ is the magnitude of the perturbation which is small enough to be imperceptible, and l is the target label. Thus the generated adversarial example x' is calculated as: $x' = x + \eta$. This perturbation can be computed simply using backpropagation. The fast gradient sign method uses the gradient of the loss function to determine in which direction the input data should be changed (whether it should be increased or decreased) to minimize the loss function.

The fast gradient sign method optimizes the networks in terms of the L_∞ distance metric. While it is fast in speed, it is not designed primarily aiming at finding the optimal adversarial examples.

Kurakin *et al.* [8] extended the fast gradient sign method by running a finer optimization (smaller change) for multiple iterations. In each iteration, pixel values are clipped to avoid large change on each pixel:

$$clip_{x,\xi}\{x'\} = \min\{255, x + \xi, \max\{0, x - \epsilon, x'\}\} \tag{7}$$

Where $clip_{x,\xi}\{x'\}$ is the clipping value in each iteration limited by ξ. The adversarial examples were generated in multiple iterations:

$$x_0 = x \tag{8a}$$

$$x_{n+1} = clip_{x,\xi}\{x_n + \epsilon sign(\nabla_x J_\theta(x_n, y))\} \tag{8b}$$

Iterative gradient sign was found to produce superior results to fast gradient sign.

To attack a specific class with enhanced capability, an alternative version of FGSM was proposed to select the least-likely class of the prediction and try to maximize the cross-entropy loss. This method is known as Iterative Least-Likely Class method:

$$x_0 = x \tag{9a}$$

$$y_{LL} = \arg\min_y \{p(y|x)\} \tag{9b}$$

$$x_{n+1} = clip_{x,\xi}\{x_n + \epsilon sign(\nabla_x J_\theta(x_n, y_{LL}))\} \tag{9c}$$

As another variation, the Target Class Gradient Sign Method (TGSM) can be extended to a more general case where the y_{LL} in 9(b) could be any desired target class.

B. JACOBIAN-BASED SALIENCY MAP ATTACK (JSMA)

Papernot *et al.* [9] designed an efficient saliency adversarial map under L_0 distance, called Jacobian-based Saliency Map Attack (JSMA). JSMA computes the Jacobian matrix of a given sample x which is expressed as:

$$J_f(x) = \frac{\partial f(x)}{\partial x} = [\frac{\partial f_j(x)}{\partial x_i}]_{i\times j} \tag{10}$$

In this way, the input features of x that made most significant changes to the output can be identified. Those most influential features are employed where some small input deviations trigger large output variations.

C. DEEPFOOL

Moosavi-Dezfooli *et al.* [10] proposed DeepFool to find the closest distance from original input to the decision boundary of adversarial examples. DeepFool is an untargeted attack technique optimized for the L_2 distance metric. An iterative attack by linear approximation is proposed in order to overcome the non-linearity in high dimension. Starting from an affine classifier, it is found that the minimal perturbation of an affine classifier is the distance to the separating affine hyperplane $\mathcal{F} = \{x : w^T x + b = 0\}$. The perturbation of an affine classifier f can be $\eta^*(x) = -\frac{f(x)}{\|w\|^2} w$.

If f is a binary differentiable classifier, an iterative method is used to approximate the perturbation by considering f is linearized around x_i at each iteration. The minimal perturbation is given by:

$$\underset{\eta_i}{\text{argmin}} \parallel \eta_i \parallel_2 \tag{11a}$$

$$s.t. f(x_i) + \nabla f(x_i)^T \eta_i = 0 \tag{11b}$$

This result can also be extended to the multi-class classifier by finding the closest hyperplane. It can also be extended to more general ℓ_p norm, $p \in [0, \infty)$.

D. CW ATTACK

Carlini and Wagner [11] invented a targeted attack to defeat defensive distillation. CW attack is effective for most of existing adversarial detecting defenses.

A new objective function g is defined so that:

$$\min_{\eta} \parallel \eta \parallel_p + c \cdot g(x + \eta) \tag{12a}$$

$$s.t. x + \eta \in [0, 1]^n \tag{12b}$$

Where $g(x') \geq 0$ if and only if $f(x') = l'$, and l' is the label of the adversarial class in targeted adversarial examples. In this way, the distance and penalty term can be better optimized. Among example objective functions g, an effective function evaluated by the authors' experiments can be:

$$g(x') = \max(\max_{i \neq l'} Z(x')_i - Z(x')_t - \kappa) \tag{13}$$

Where κ is a constant to control the confidence.

A new variant w was proposed to avoid the box constraint, where w satisfies $\eta = \frac{1}{2}(\tanh(w) + 1) - x$. General optimizers in deep learning were employed to produce adversarial examples and conducted 20 iterations to reach an optimal c by binary searching. However, they found that if the gradients of $\parallel \eta \parallel_p$ and $g(x + \eta)$ are not in the same scale, it is hard to find a suitable constant c in all of the iterations of the gradient search and then get the optimal result. Due to this reason, two of their proposed functions did not find optimal solutions for adversarial examples.

The authors proposed ℓ_2 attack which can be given by:

$$\min_{w} \parallel \frac{1}{2}(\tanh(w) + 1) \parallel_2 + c \cdot g(\frac{1}{2}(\tanh(w) + 1)) \tag{14}$$

ℓ_∞ attack was also an iterative attack, which replaced the ℓ_2 term with a new penalty in each iteration:

$$\min c \cdot g(x + \eta) + \sum_i [(\eta_i - \tau)^+] \tag{15}$$

For each iteration, they reduced τ by a factor of 0.9, if all $\eta_i < \tau$. ℓ_∞ attack considered τ as an approximate measurement of ℓ_∞.

V. EVALUATION METHODOLOGY

A. NSL-KDD DATASET

One of the most used dataset to test intrusion detection algorithms is the KDD'99 dataset [13] which was used from the DARPA'98 IDS evaluation program. Researchers identified two major drawbacks with the KDD'99 dataset [12]: an enormous amount of redundant records are found both in the training and test data; some classes of attacks are too readily to detect due to dataset imbalance. The NSL-KDD dataset [14], which was an improved version of the KDD'99 dataset, was proposed to overcome the limitation of the KDD'99 dataset in two ways: all the redundant records from the training and test data are removed; the records in the KDD'99 dataset are rebalanced according to their difficulty levels of classification, making it more reasonable and realistic for benchmarking learning algorithms.

Each record in the NSL-KDD dataset has 41 features. The detailed list of features is presented in Table 1. The features belong to three major families [12]:

- *Basic features* are the ones related to connection information such as hosts, ports, services used and protocols.
- *Traffic features* are the ones that are calculated as an aggregate during a window interval. These are further categorized as aggregates based on the same host and aggregates over the same service. A notable difference between KDD'99 and NSL-KDD dataset is that in the latter, the time window was substituted with a connection window of the last 100 connections.
- *Content features* are extracted from the packet data or payload and they are related to the content of specific applications or the protocols used.

Each record in the NSL-KDD dataset is labeled with either normal or a particular class of attack. The training data contains 23 traffic classes that include 22 classes of attack and one normal class. The test data contains 38 traffic classes that include 21 attacks classes from the training data, 16 novel attacks, and one normal class.

TABLE 1. List of the NSL-KDD dataset.

No.	Feature	Type	Description
	Basic features of individual TCP connections		
1	Duration	Numeric	Duration of the connection
2	Protocol_type	Nominal	Type of the protocol
3	Service	Nominal	Network service on the destination
4	Flag	Nominal	Normal or error status of the connection
5	Src_bytes	Numeric	# of bytes transferred from source to destination
6	Dst_bytes	Numeric	# of bytes transferred from destination to source
7	Land	Binary	1 if connection is from/to the same host/port; 0 otherwise
8	Wrong_fragment	Numeric	# of "wrong" fragments
9	Urgent	Numeric	# of urgent packets (with the urgent bit set)
	Content features within a connection suggested by domain knowledge		
10	Hot	Numeric	# of "hot" indicators
11	Num_failed_logins	Numeric	# of failed login attempts
12	Logged_in	Binary	1 if successfully logged in; 0 otherwise
13	Num_compromised	Numeric	# of "compromised" conditions
14	Root_shell	Binary	1 if root shell is obtained; 0 otherwise
15	Su_attempted	Binary	1 if "su root" command attempted; 0 otherwise
16	Num_root	Numeric	# of "root" accesses
17	Num_file_creations	Numeric	# of file creation operations
18	Num_shells	Numeric	# of shell prompts
19	Num_access_files	Numeric	# of operations on access control files
20	Num_outbound_cmds	Numeric	# of outbound commands in an ftp session
21	Is_hot_login	Binary	1 if the login belongs to the "hot" list; 0 otherwise
22	Is_guest_login	Binary	1 if the login is a "guest" login; 0 otherwise
	Traffic features computed using a two-second time window		
23	Count	Numeric	# of connections to the same host as the current connection (*Note: The following features refer to these same-host connections.*)
24	Serror_rate	Numeric	# of connections that have "SYN" errors
25	Rerror_rate	Numeric	% of connections that have "REJ" errors
26	Same_srv_rate	Numeric	% of connections to the same service
27	Diff_srv_rate	Numeric	% of connections to different services
28	Srv_count	Numeric	% of connections to the same service as the current connection in the past two seconds (*Note: The following features refer to these same-service connections.*)
29	Srv_serror_rate	Numeric	% of connections that have "SYN" errors
30	Srv_rerror_rate	Numeric	% of connections that have "REJ" errors
31	Srv_diff_host_rate	Numeric	% of connections to different hosts
	Host based traffic features computed using a two-second time window		
32	Dst_host_count	Numeric	# of connections having the same destination host
33	Dst_host_srv_count	Numeric	# of connections using the same service
34	Dst_host_same_srv_rate	Numeric	% of connections using the same service
35	Dst_host_srv_diff_host_rate	Numeric	% of different services on the current host
36	Dst_host_same_src_port_rate	Numeric	% of connections to the current host having the same src port
37	Dst_host_srv_diff_host_rate	Numeric	% of connections to the same service coming from different hosts
38	Dst_host_serror_rate	Numeric	% of connections to the current host that have an S0 error
39	Dst_host_srv_serror_rate	Numeric	% of connections to the current host that and specified service that have an S0 error
40	Dst_host_rerror_rate	Numeric	% of connections to the current host that have an RST error
41	Dst_host_srv_rerror_rate	Numeric	% of connections to the current host and specified service that have an RST error

B. NSL-KDD DATASET

1) ONE-HOT ENCODING

The features in the NSL-KDD dataset have three data types: nominal, binary, and numeric. Binary data can be viewed as variables that contain numeric values since a numeric value is enough to indicate the presence (1) or absence (0) of a specific status. Nominal data are variables that contain categorical values rather than numeric values. Many machine learning algorithms including neural networks cannot operate on nominal data directly. So we use one-hot encoding to convert nominal features to numeric feature. In the NSL-KDD dataset, there are three nominal features: "protocol_type", "service", and "flag". We take the feature "protocol_type" as an example. It has three categorical values: "tcp", "udp", and "icmp". By one-hot encoding, three new numeric features are created to replace the original feature "protocol_type": "protocol_type_tcp", "protocol_type_udp", and "protocol_type_icmp". The binary value for each new feature is an indicator of that corresponding protocol type's presence. Of the three new columns produced from "protocol_type", only one could take on the value 1 for each sample. For example, the list of feature "protocol_type" for four samples [tcp, udp, icmp, udp] becomes [[1,0,0,0],[0,1,0,1],[0,0,1,0]] in the one-hot

encoded form. Using one-hot encoding, the feature "service" is transformed to 70 new features, and the feature "flag" to 11 new features. In this way, the 41-feature dataset is mapped to a 122-feature dataset.

2) NORMALIZATION

After numericalization using one-hot encoding, the dataset consists of numeric features whose values can be drawn from different distributions, have different scales and, sometimes, contaminated by outliers. If there are big differences in the

TABLE 2. Classification of attack types.

Attack Label	Attack Type
Denial of Service (DOS)	Back, Land, Neptune, Pod, Smurf, Teardrop, Apache2, Udpstorm, Processtable, Worm, Mailbomb
Probe	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm
Remote to Local (R2L)	Guess_Password, Ftp_write, Imap, Phf, Multihop, Warezmaster, Warezclient, Spy, Xlock, Xsnoop, Snmpguess, Snmpgetattack, ProbHttptunnel, Sendmail, Named
User To Root (U2R)	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps

ranges of different features and no outliers, features with very large values may cause imbalanced results by some classifiers. So we simply apply the min-max scaling to each feature column, where the new normalized value x_{norm} can be calculated as follows:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{16}$$

Here, x is a particular sample, x_{min} is the smallest value in a feature column, and x_{max} the largest value, respectively. The rescaling maps the features to a range of [0, 1].

3) CLASSIFICATION OF ATTACK TYPES

The attack types in the NSL-KDD dataset are classified into four major families: Denial of Service (DOS), Probe, Remote to Local (R2L), and User to Root (U2R) attacks:

- *DOS attacks* are attacks that target availability or prevent legitimate users from accessing information or services.
- *Probe attacks* are attacks that aim at gathering information by scanning or probing the network.
- *R2L attacks* are attacks that attempt to gain unauthorized remote access to a local machine.
- *U2R attacks* are attacks that attempt to access normal user account and exploit vulnerabilities in the system for privilege escalation.

The detailed list of classified attack types is presented in Table 2. After the classification, the 39 attack types are transformed to the 4 attack labels.

4) PRE-PROCESSED DATASET SUMMARY

After one-hot encoding, normalization, and classification of attack types, the problem was transformed to a 5-class classification problem where the 5 labels are ''Normal'', ''DOS'', ''Probe'', ''R2L'', and ''U2R'', and the 122 numeric features fall into the range between 0 and 1. The dataset have the training set and the test sets. The number of samples in the training set is 125,973 and in the test set 22,544.

C. METHODOLOGY

Given that the number of samples in the dataset can be considered sufficient, we use the simple holdout cross-validation method to assess the model performance. Specifically, we split the original training data into the training set which account for 90% of the original training set and the validation set which for 10%. That is, the number of the sample in the training set is 113,375 and in the validation set 12,598.

Similar to most existing deep learning research, our models and attack algorithms were implemented using TensorFlow [15]. The experimental results were parsed and analyzed in Python. All of our evaluations were performed on a personal desktop without GPU acceleration.

In this paper, we only consider white-box attacks where the target deep neural networks are known by the adversary. We use multilayer perceptrons (MLPs) as the neural network architecture for intrusion detection. The MLPs are constructed with two hidden layers and each layer contains 256 neural units. The activation function of each hidden unit is Rectified Linear Unit (ReLU). For regularization, a dropout layer with the dropout rate of 0.4 is adopted after each hidden layer. The dropout layers are applied to control over-fitting by removing an individual unit with an arbitrary probability while training the network. A softmax layer is employed after the logits layer as the output of the classifier. The optimizer, the batch size, and the learning rate used in training the networks is ADAM, 128, and 0.001 respectively. The cross-entropy cost function is used as the loss function to be minimized by training.

First, we feed the training dataset to the deep neural networks and allow enough epochs to obtain the well-trained deep neural networks. The deep neural networks are used as the target of attacks as well as the baseline of our evaluations. Then we implement the four attack algorithms and use them respectively to generate the adversarial examples from the test dataset based on the deep neural networks. Finally, we evaluate the performance of the classifier using both the test dataset and the adversarial examples.

D. METRICS

The performance evaluation is conducted based on the following metrics:

- True Positive (TP) - Attack data that is correctly classified as an attack.
- False Positive (FP) - Normal data that is incorrectly classified as an attack.
- True Negative (TN) - Normal data that is correctly classified as normal.

- False Negative (FN) - Attack data that is incorrectly classified as normal.

The confusion matrix of a binary classifier is defined in Table 3, which can be generalized to the confusion matrix of a multi-class classifier.

TABLE 3. Confusion matrix of a binary classifier.

Actual Label \ Predicted Label	Anomaly	Normal
Anomaly	TP	FN
Normal	FP	TN

The following measures are used to evaluate the performance of the classifier:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{17}$$

The accuracy measures the proportion of the total number of correct classifications.

$$Precision = \frac{TP}{TP + FP} \tag{18}$$

The precision measures the number of correct classifications penalized by the number of incorrect classifications.

$$Recall = \frac{TP}{TP + FN} \tag{19}$$

The recall measures the number of correct classifications penalized by the number of missed entries.

$$FalseAlarm = \frac{FP}{FP + TN} \tag{20}$$

The false alarm measures the proportion of benign events incorrectly classified as malicious.

$$F - score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{21}$$

The F-score measures the harmonic mean of precision and recall, which serves as a derived effectiveness measurement.

VI. RESULTS AND DISCUSSION

A. CLEAN DATA

We start with the performance of deep neural networks whose input is the clean data. The count of samples belonging to each label is summarized in Table 4. The metrics for the MLP classifier on the clean dataset are presented in Table 5.

ROC curves are typically used in binary classification to study the output of a classifier. In order to extend ROC curve and ROC area to multi-class or multi-label classification, it is necessary to binarize the output. One ROC curve can be drawn per label, but one can also draw a ROC curve by considering each element of the label indicator matrix as a binary prediction (micro-averaging). Another evaluation measure for multi-class classification is macro-averaging,

TABLE 4. Number of samples in the training and test set.

Label	No. of Training Samples	No. of Test Samples
Normal	67343	9711
DOS	45927	7460
Probe	11656	2421
R2L	995	2885
U2R	52	67

TABLE 5. Metrics for the MLP classifier on the clean dataset.

Label	Accuracy	Precision	Recall	False Alarm	F-score
Normal	78.19	67.12	96.81	35.94	79.25
DOS	92.50	96.19	80.56	1.59	87.69
Probe	95.25	83.62	69.50	1.64	75.88
R2L	88.25	98.38	8.39	0.02	15.46
U2R	99.75	89.50	25.38	0.01	39.53

which gives equal weight to the classification of each label. The ROC curves for the MLP classifier on the clean dataset are shown in Fig. 1.

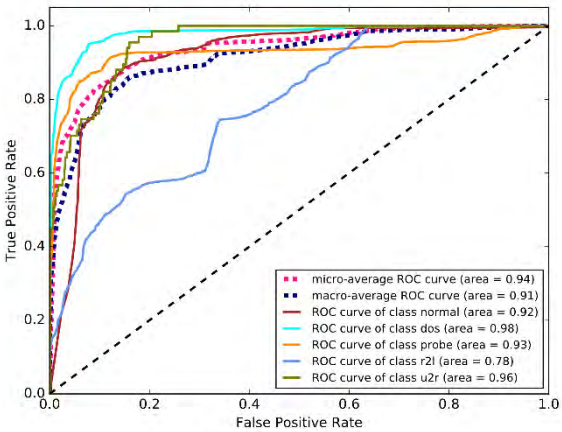

FIGURE 1. ROC curves for the MLP classifier on the clean dataset.

B. JSMA ATTACKS

1) ATTACKS FROM SCRATCH

We first investigate the performance of JSMA attacks from scratch. During each iteration, the JSMA method finds the feature that has the most influence on the result (most salient feature) and add noise to the feature. We set the maximum epochs to 100 and the noise added to input per epoch to 0.5. The minimum and maximum values in output tensor are set to 0.0 and 1.0 respectively. We let the original sample be a single

TABLE 6. Adversarial samples produced by JSMA from scratch.

Target Label	Probability	Altered Features (L_0)
Normal	1.0	Duration, Wrong_fragment, Num_compromised, Su_attempted, Num_root, Num_access_files, Srv_count, Diff_srv_rate, Service_IRC ($L_0 = 9$)
DOS	1.0	Duration, Wrong_fragment, Num_compromised, Su_attempted, Num_root, Num_access_files, Srv_count, Diff_srv_rate, Service_IRC ($L_0 = 9$)
Probe	1.0	Duration, Wrong_fragment, Num_compromised, Su_attempted, Num_root, Num_access_files, Srv_count, Diff_srv_rate, Service_IRC ($L_0 = 9$)
R2L	1.0	Duration, Wrong_fragment, Num_compromised, Su_attempted, Num_root, Num_access_files, Srv_count, Diff_srv_rate, Service_IRC ($L_0 = 9$)
U2R	0.44	Duration, Wrong_fragment, Num_compromised, Su_attempted, Num_root, Num_access_files, Srv_count, Diff_srv_rate, Service_IRC ($L_0 = 9$)

TABLE 7. Metrics for the MLP classifier on the JSMA-generated dataset.

Label	Accuracy	Precision	Recall	False Alarm	F-score
Normal	52.41	42.72	30.64	31.11	35.69
DOS	60.31	33.88	20.95	20.23	25.89
Probe	74.06	10.84	19.58	19.38	13.95
R2L	67.00	12.74	27.00	27.12	17.31
U2R	97.62	0.42	2.98	2.10	0.74

sample whose features are all set to zero. The adversarial samples produced by JSMA for each target label have their probability and altered features listed in Table 6. We can see that only 9 out of the 122 features are needed to ensure 100% probability of misguiding the deep neural networks to the four target classes: ''Normal'', ''DOS'', ''Probe'', and ''R2L''. However, it is not so easy to fool the deep neural networks into the ''U2R'' class as the success rate is 0.44 with the minimum L_0 constraint. We find that three basic features (namely ''Duration'', ''Wrong_fragment'', and ''Service_IRC''), four content features (namely Num_compromised, Su_attempted, Num_root, and Num_access_files), and two traffic features (''Srv_count'' and ''Diff_srv_rate'') count for JSMA attacks. Note that no host based traffic features are exploited and that seems to indicate that host based traffic features weight less in the deep neural network based classifiers.

2) ATTACKS FROM ORIGINAL SAMPLES

We set the test set as the original samples and use the JSMA algorithm to generate the adversarial samples. To evaluate the overall performance of the JSMA attacks, we choose a random target label for each adversarial sample. During each iteration, the JSMA method finds the feature that has the most influence on the result (most salient feature) and add noise to the feature. We set the maximum epochs to 30 and the noise

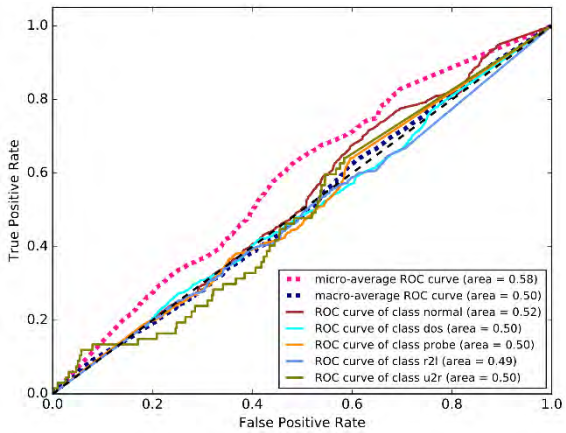

FIGURE 2. ROC Curves for the MLP classifier on the JSMA-generated dataset.

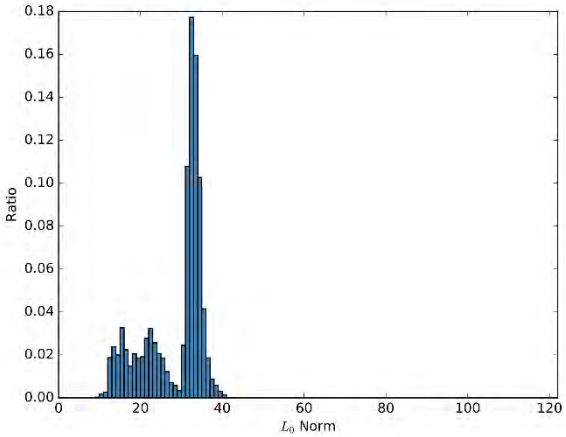

FIGURE 3. L_0 norm for the JSMA-generated dataset.

added to input per epoch to 1.0. The minimum and maximum values in output tensor are set to 0.0 and 1.0 respectively.

The metrics for the MLP classifier on the JSMA-generated dataset are presented in Table 7. We can see that JSMA attacks successfully degrade the performance of MLP classifier.

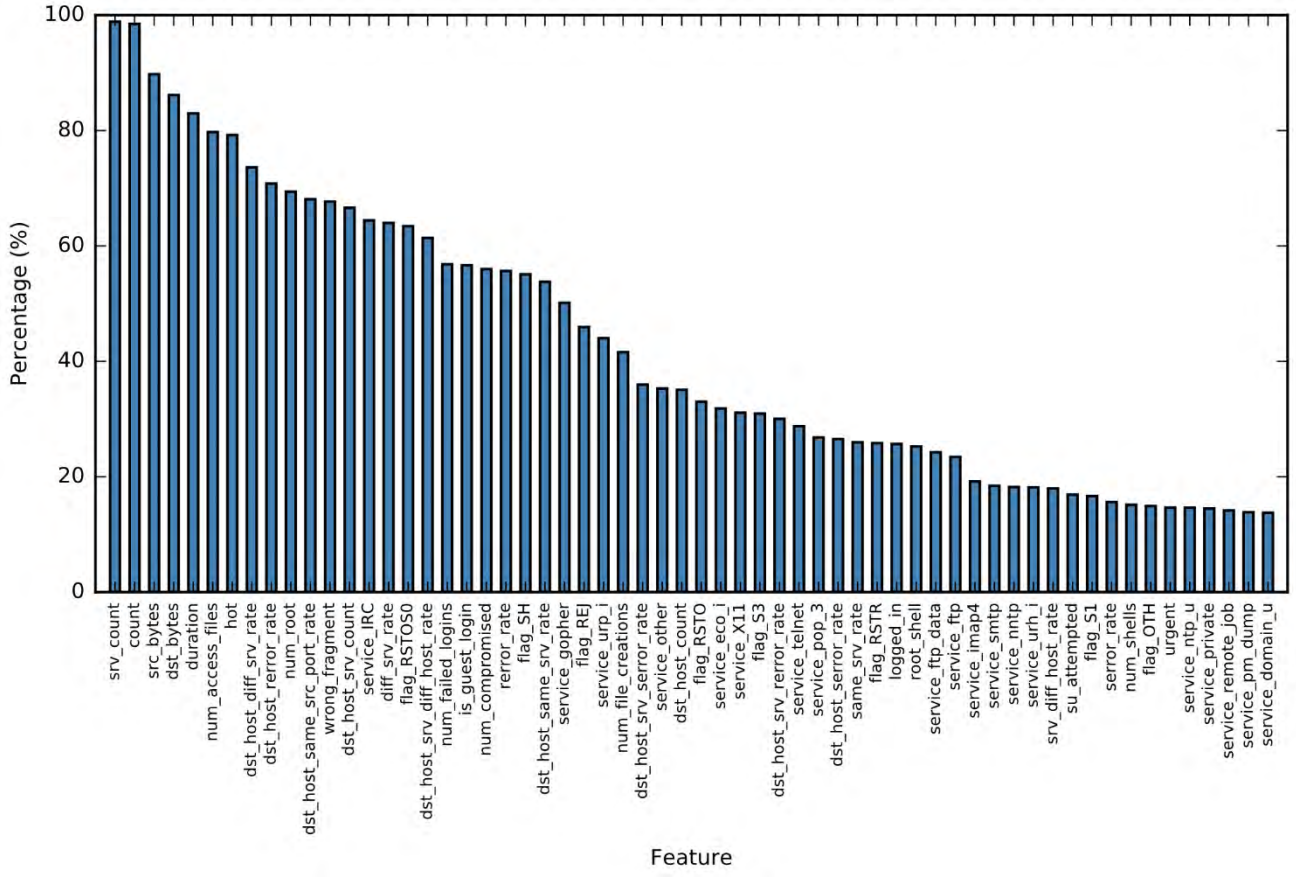

FIGURE 4. Top 60 features altered for the JSMA-generated dataset.

The ROC curves for the MLP classifier on the JSMA-generated dataset are shown in Fig. 2. The values of AUC (Area Under Curve) for all classes are suppressed to around 0.5 under JSMA attacks. The distribution of the L_0 norm is shown in Fig. 3. The mean L_0 norm is 27.36 while the number of unique features changed over all samples is 119. The top 60 features that are frequently chosen by adversarial examples are depicted in Fig. 4. The hit rate in Fig. 4 is highly skewed towards a small set of features. Note that 18 out of the top 60 features are virtually derived from one original feature ''service'' (before being one-hot encoded).

C. FGSM ATTACKS

1) UNTARGETED ATTACKS

We first consider the untargeted version of the FGSM attacks. The scale factor for noise is set to 0.02. The maximum epoch is set to 12. We use gradient sign to generate the adversarial examples. The minimum and maximum values in output tensor are set to 0.0 and 1.0 respectively.

The metrics for the MLP classifier on the untargeted FGSM-generated dataset are presented in Table 8. A great performance degrading can be found in Table 8 compared with the clean dataset shown in Table 5. Fig. 5 illustrates the ROC curves for the MLP classifier on the untargeted

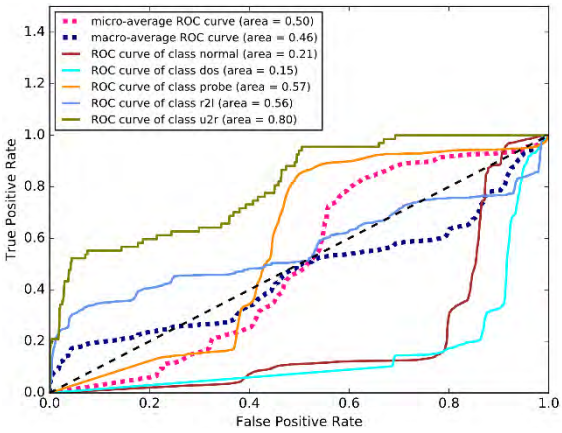

FIGURE 5. ROC curves for the MLP classifier on the untargeted FGSM-generated dataset.

FGSM-generated dataset. We can find in Fig. 5 that the values of AUC for class normal and class dos are even further decreased to 0.21 and 0.15 respectively while average AUC and the AUCs for the remaining classes stay at around 0.5. The L_∞ for which the MLP classifier are optimized by the FGSM attacks has its mean as 0.2401 and its variance as 0.00015. That result means that all samples are evenly

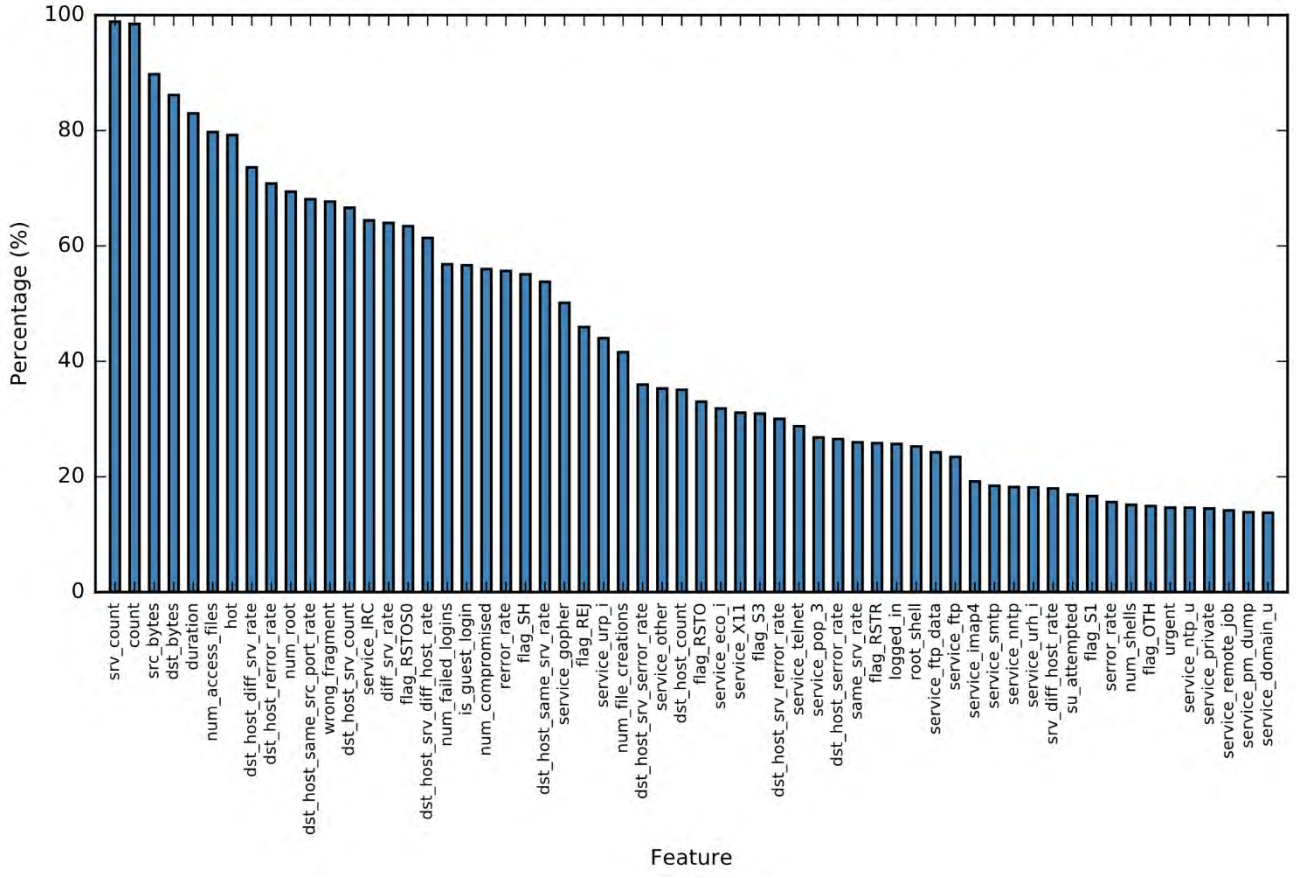

FIGURE 6. Top 60 features altered for the untargeted FGSM-generated dataset.

perturbed by the untargeted FGSM attacks in terms of L_∞. The number of unique features changed is 122 and the number of average features changed per datapoint is 74.80. The top 60 features that are frequently chosen by adversarial examples are presented in Fig. 6. We find that 45 out of the top 60 features are those generated by one-hot encoding from the original feature "service". That indicates a greater weight of feature "service" in generating adversarial examples by the untargeted FGSM attacks than by the JSMA attacks shown in Fig. 4. Besides, the choice rate of top features are more evenly distributed in Fig. 6 than in Fig. 4.

2) LEAST-LIKELY ATTACKS

For the targeted FGSM attacks, we first examine the attacks which set the desired target label to the least-likely class. The scale factor for noise is set to 0.02. The maximum epoch is set to 12. We use gradient sign to generate the adversarial examples. The minimum and maximum values in output tensor are set to 0.0 and 1.0 respectively.

The metrics for the MLP classifier on the least-likely targeted FGSM-generated dataset are presented in Table 9. Fig. 7 illustrates the ROC curves for the MLP classifier on the least-likely targeted FGSM-generated dataset. Comparing Fig. 7 against Fig. 5, we can see the least-likely targeted

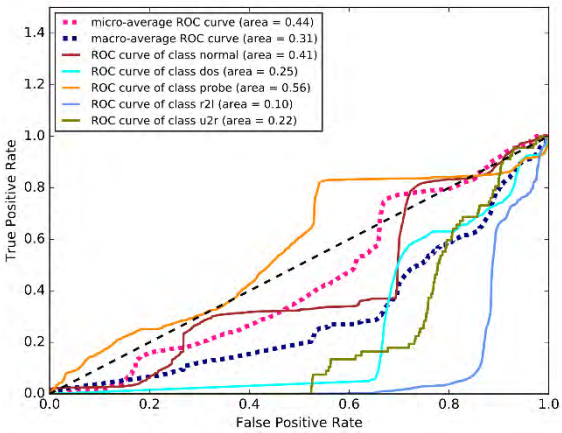

FIGURE 7. ROC curves for the MLP classifier on the least-likely targeted FGSM-generated dataset.

FGSM attacks demonstrate greater overall adversarial power than the untargeted FGSM attacks in terms of the average AUC. The L_∞ optimized by the least-likely targeted FGSM attacks has its mean as 0.240 and its variance as 0.00013 both of which are almost equivalent to those of the untargeted FGSM attacks. That result means that all samples are evenly perturbed by the untargeted FGSM attacks in terms of L_∞.

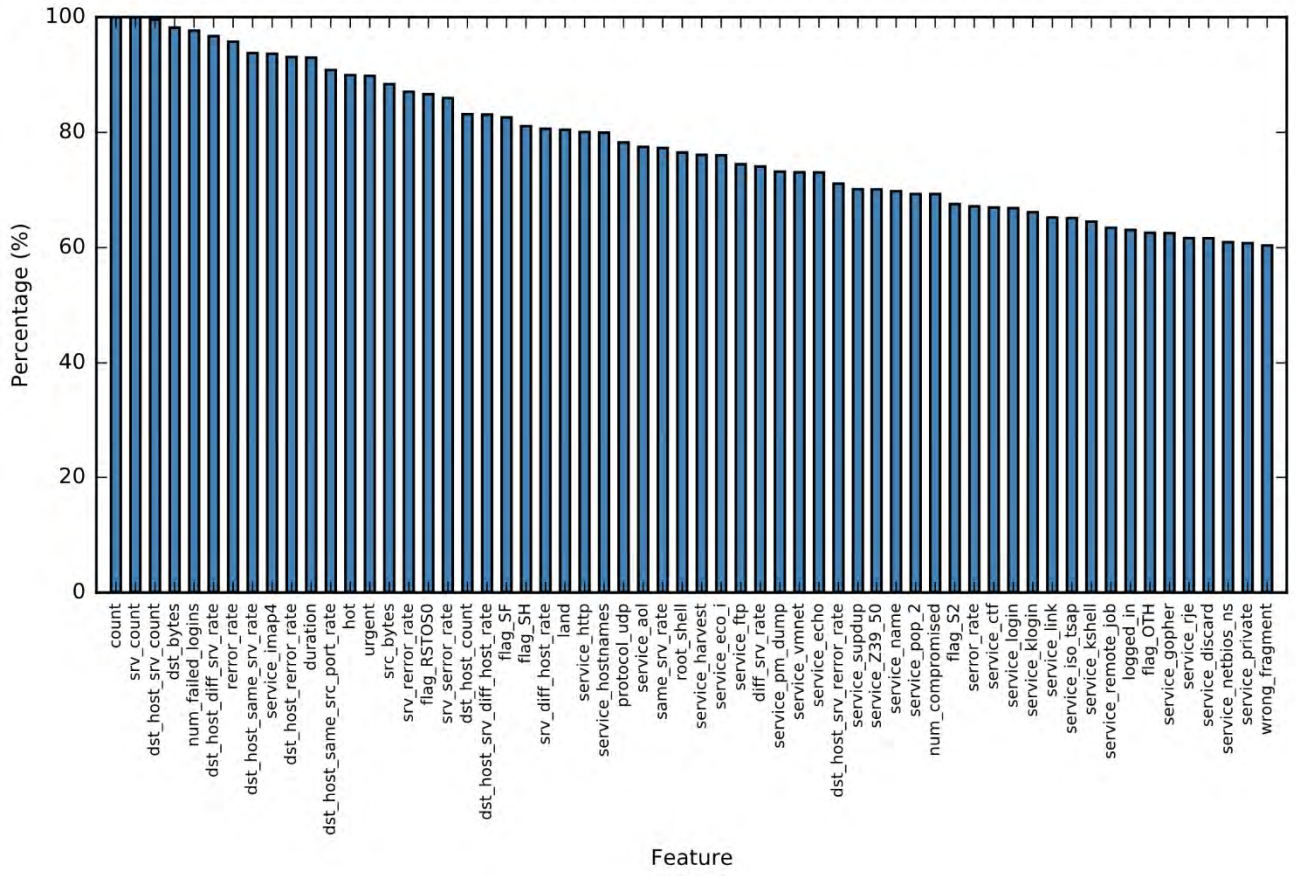

FIGURE 8. Top 60 features altered for the least-likely targeted FGSM-generated dataset.

The number of unique features changed is 122 and the number of average features changed per datapoint is 76.85. They are also very close to or even the same as those of the untargeted FGSM attacks. The top 60 features that are frequently chosen by adversarial examples are presented in Fig. 8. The features listed in Fig. 8 and those in Fig. 6 have much overlap.

3) RANDOM-TARGET ATTACKS

We then consider the attacks which randomly select the desired target label. The scale factor for noise is set to 0.02.

The maximum epoch is set to 8. We use gradient sign to generate the adversarial examples. The minimum and maximum values in output tensor are set to 0.0 and 1.0 respectively.

The metrics for the MLP classifier on the random-targeted FGSM-generated dataset are presented in Table 10. Fig. 9 illustrates the ROC curves for the MLP classifier on the random-targeted FGSM-generated dataset. Comparing Fig. 9 against Fig. 7, we can identify obvious better performance of the MLP classifier in Fig. 9 than that in Fig. 7 since all AUC values in Fig. 9 are improved. That is because a random target label is not always the least-likely label which is the best target label used to fool the classifier. The L_∞ optimized by the random-targeted FGSM attacks has its mean

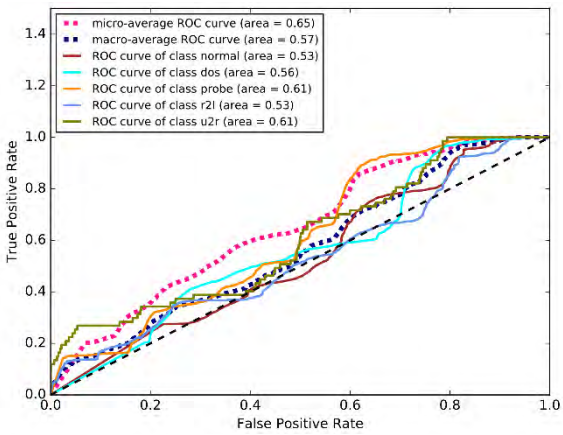

FIGURE 9. ROC curves for the MLP classifier on the random-target FGSM-generated dataset.

as 0.16 and its variance as 0.00016. The number of unique features changed is 122 and the number of average features changed per datapoint is 76.70.

They are also very close to or even the same as those of both the untargeted and the least-likely targeted FGSM attacks. The top 60 features that are frequently chosen by adversarial examples are presented in Fig. 10. The features

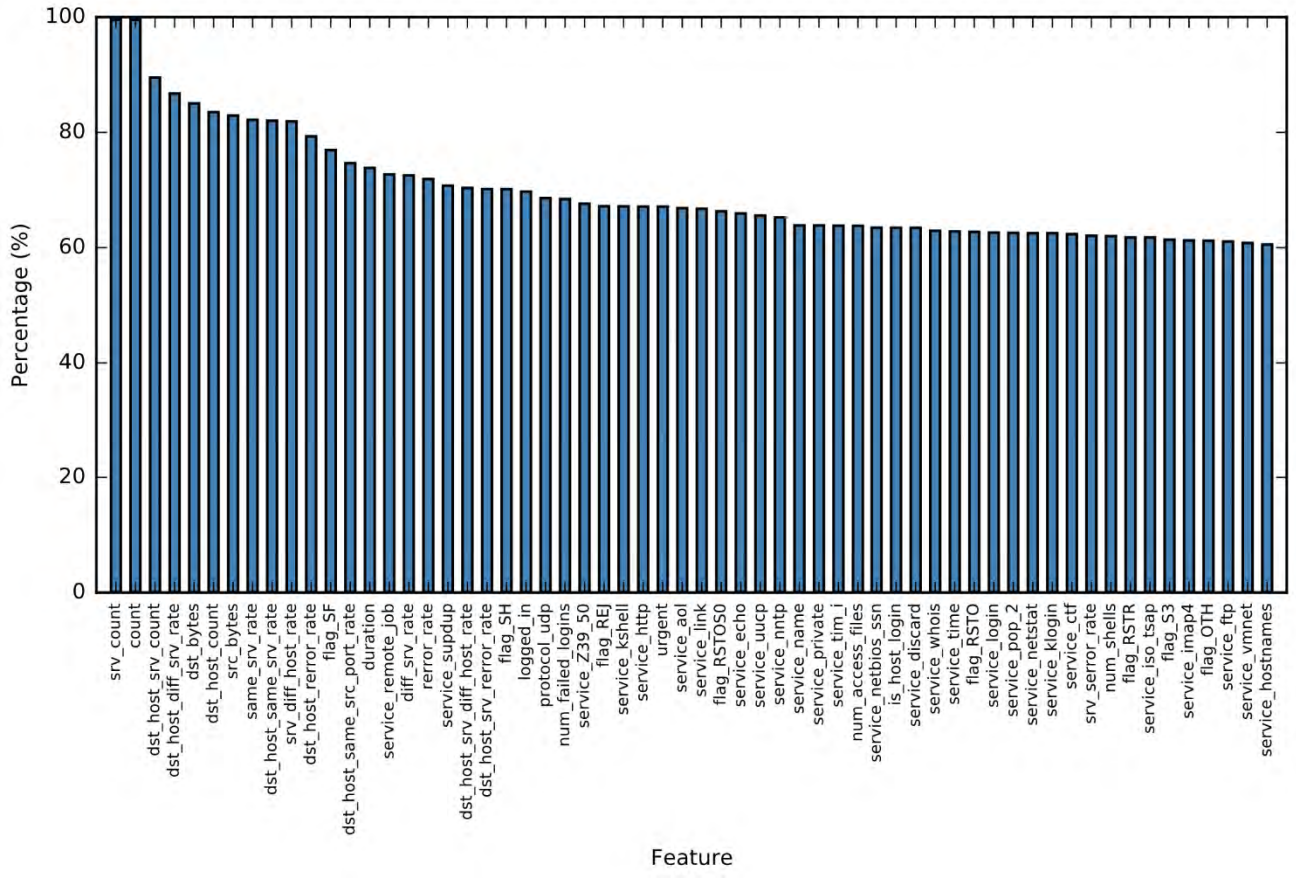

FIGURE 10. Top 60 features altered for the random-target FGSM-generated dataset.

TABLE 8. Metrics for the MLP classifier on the untargeted FGSM-generated dataset.

Label	Accuracy	Precision	Recall	False Alarm	F-score
Normal	40.78	6.83	2.96	30.61	4.14
DOS	24.48	6.95	10.35	68.56	8.31
Probe	66.94	6.15	14.58	26.78	8.65
R2L	87.25	50.38	25.78	3.73	34.12
U2R	99.69	NA	0.00	0.00	NA

TABLE 9. Metrics for the MLP classifier on the least-likely targeted FGSM-generated dataset.

Label	Accuracy	Precision	Recall	False Alarm	F-score
Normal	50.66	40.38	30.55	34.16	34.78
DOS	25.34	4.73	6.57	65.38	5.50
Probe	84.69	15.77	9.88	6.35	12.14
R2L	72.94	0.03	0.03	16.38	0.03
U2R	99.19	0.00	0.00	0.49	NA

listed in Fig. 8 and those in both Fig. 6 and Fig. 10 have much overlap.

D. DEEPFOOL ATTACKS

In our evaluation of DeepFool attacks, we set the small overshoot value to cross the boundary to 0.01. The maximum epoch is set to 3. The minimum and maximum values in output tensor are set to 0.0 and 1.0 respectively. The minimum probability for adversarial samples is set to 0.

The metrics for the MLP classifier on the DeepFool-generated dataset are shown in Table 11. Fig. 11 illustrates the ROC curves for the MLP classifier on the DeepFool-generated dataset. In Fig. 11, the AUCs for both class normal and class dos are very small compared with the average AUC and the AUCs for the other classes. The L_2 optimized by the DeepFool attacks has its mean as 0.773 and its variance as 0.0376. Its distribution is shown in Fig. 12 The number of unique features changed is 122 and the number of average

TABLE 10. Metrics for the MLP classifier on the random-target FGSM-generated dataset.

Label	Accuracy	Precision	Recall	False Alarm	F-score
Normal	50.66	42.84	43.53	43.91	43.19
DOS	61.41	37.44	24.88	20.55	29.89
Probe	75.75	13.05	22.27	17.83	16.45
R2L	77.19	16.56	19.38	14.33	17.86
U2R	98.81	3.60	11.94	0.95	5.54

TABLE 11. Metrics for the MLP classifier on the deepfool-generated dataset.

Label	Accuracy	Precision	Recall	False Alarm	F-score
Normal	41.03	7.12	3.06	30.22	4.28
DOS	17.33	8.93	16.28	82.19	11.53
Probe	71.62	5.90	10.98	21.09	7.68
R2L	87.25	53.00	4.30	0.56	7.95
U2R	99.69	47.06	11.94	0.04	19.05

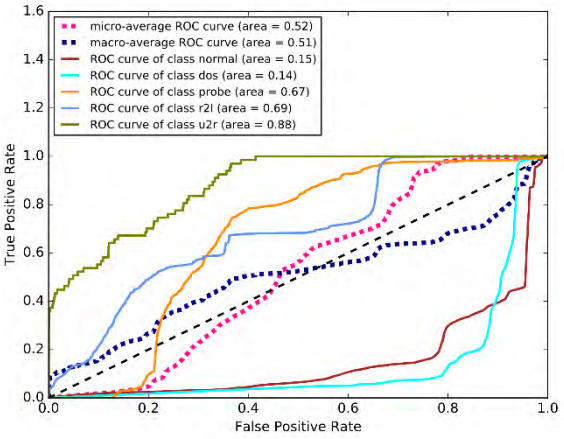

FIGURE 11. ROC curves for the MLP classifier on the DeepFool-generated dataset.

features changed per datapoint is 62.31. The top 60 features that are frequently chosen by adversarial examples are presented in Fig. 13.

E. CW ATTACKS

In our evaluations of CW Attacks, we set the scaling factor for the second penalty term to 3.0. The $p-norm$ is set to 2. The temperature for sigmoid function is set to 2. We use the Adam optimizer with learning rate 0.1 to minimize the CW loss. The minimum confidence of adversarial examples is set to 0. The maximum epoch is set to 100. To facilitate the evaluation of overall performance of the CW attacks, we randomly choose a target label for each adversarial example.

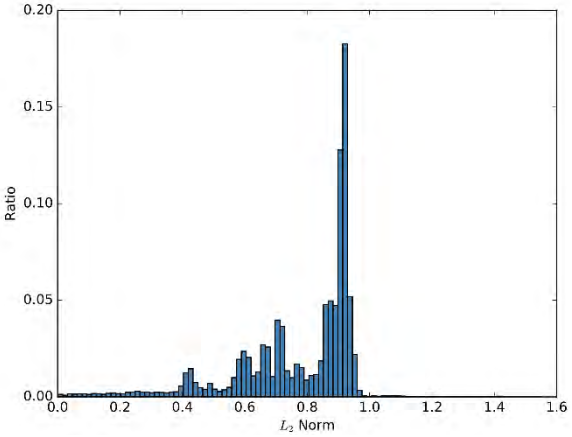

FIGURE 12. L_2 norm for the DeepFool-generated dataset.

L_0 Norm:

For the L_0-norm CW attacks, we set the decreasing factor for the upper bound of noise to 0.9.

The metrics for the MLP classifier on the L_0-norm CW-generated dataset are shown in Table 12. Fig. 14 illustrates the ROC curves for the MLP classifier on the L_0-norm CW-generated dataset. Table 12 and Fig. 14 show limited adverse impacts of the L_0-norm CW attacks on the MLP classifier, especially compared with other attacks discussed above. The L_0 optimized by the CW attacks has its mean as 115.10 and its variance as 1.655. Its distribution is shown in Fig. 15. The number of unique features changed is 122 and the number of average features changed per datapoint is 115.10. The top 60 features are chosen by all adversarial examples.

L_2 Norm:

The metrics for the MLP classifier on the L_2-norm CW-generated dataset are shown in Table 13. Fig. 16 illustrates the ROC curves for the MLP classifier on the L_2-norm CW-generated dataset. Table 13 and Fig. 16 indicate similar performance of the L_2-norm CW attacks with the L_0-norm CW attacks. The L_2 optimized by the CW attacks has its mean as 1.09 and its variance as 0.499. Its distribution is shown in Fig. 17. The number of unique features changed is 122 and the number of average features changed per datapoint is 115.10. The top 60 features are chosen by all adversarial examples.

L_∞ Norm:

The metrics for the MLP classifier on the L_∞-norm CW-generated dataset are shown in Table 14. Fig. 18 illustrates the ROC curves for the MLP classifier on the L_∞-norm CW-generated dataset. The L_∞ optimized by the CW attacks has its mean as 0.70 and its variance as 0.161. Its distribution is shown in Fig. 19. The number of unique features changed is 122 and the number of average features changed per datapoint is 115.10. The top 60 features are chosen by all adversarial examples.

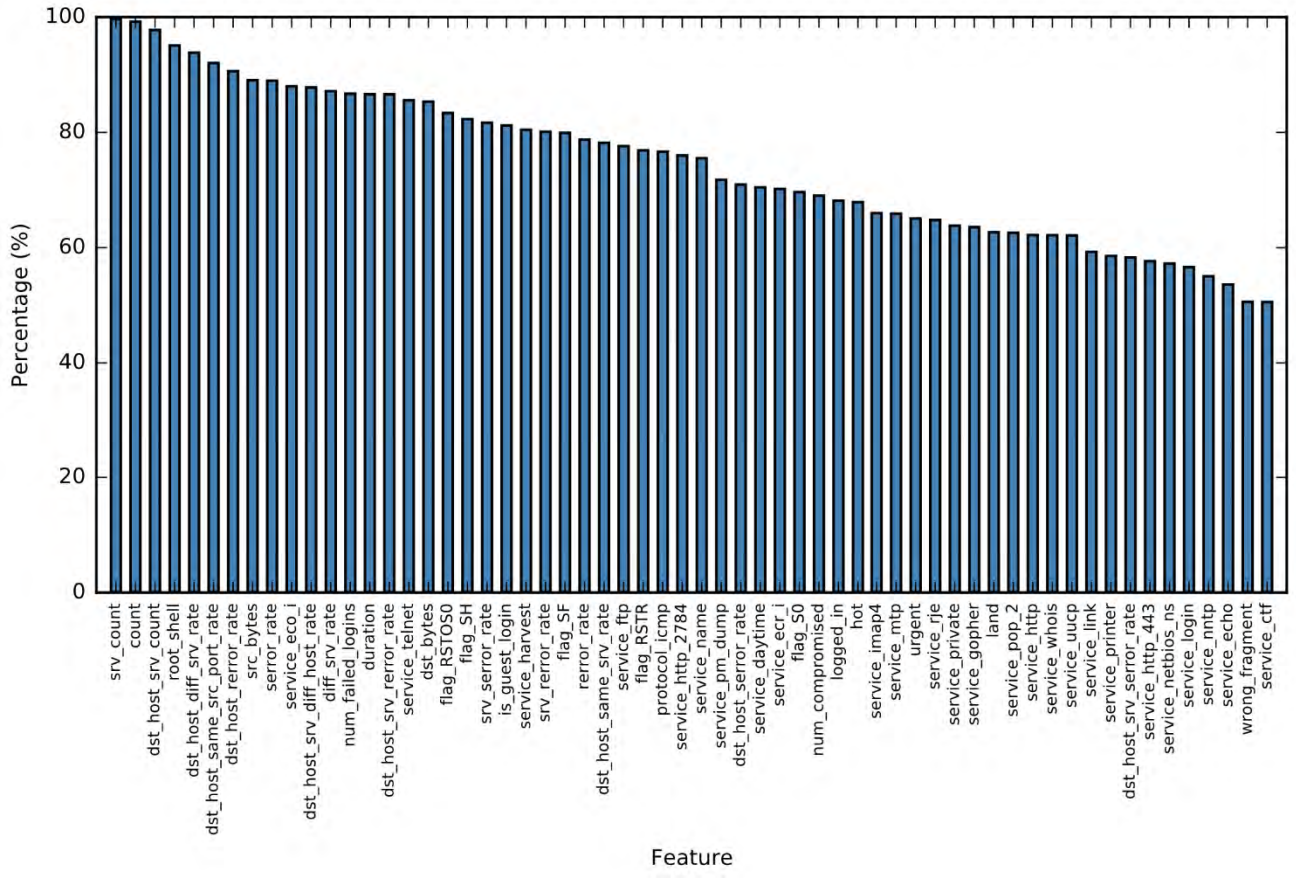

FIGURE 13. Top 60 features altered for the DeepFool-generated dataset.

TABLE 12. Metrics for the MLP classifier on the L_0-norm CW-generated dataset.

Label	Accuracy	Precision	Recall	False Alarm	F-score
Normal	64.88	56.28	82.88	48.72	67.00
DOS	70.00	58.31	32.56	11.51	41.78
Probe	83.31	27.84	34.75	10.83	30.91
R2L	86.56	43.25	15.42	2.97	22.73
U2R	99.62	33.34	16.42	0.10	22.00

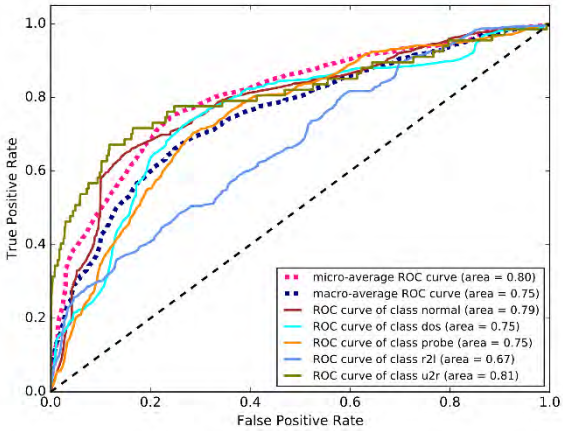

FIGURE 14. ROC curves for the MLP classifier on the L_0-norm CW-generated dataset.

F. DISCUSSION

Besides the detailed analysis above, we can reach the following key findings when comparing the results among all attacks examined:

- CW attacks seem to be less devastating than the other three attacks. However, it was reported to be more robust against some state-of-the-art defenses.
- The use of features in generating adversarial examples are comparatively more imbalanced by JSMA attacks than by the other three attacks. In particular, CW attacks tend to indiscriminately use all features even if they are optimized for L_2. Consider the fact that it would generally be easy for attackers to manipulate a small subset of features than a large one. In this way, JSMA attacks are more attractive for attackers.
- To illustrate the most used features across different attacks, we compute the intersections of all combinations of at least two attacks in Table 15. As the CW attacks do not select features, we do not show the CW

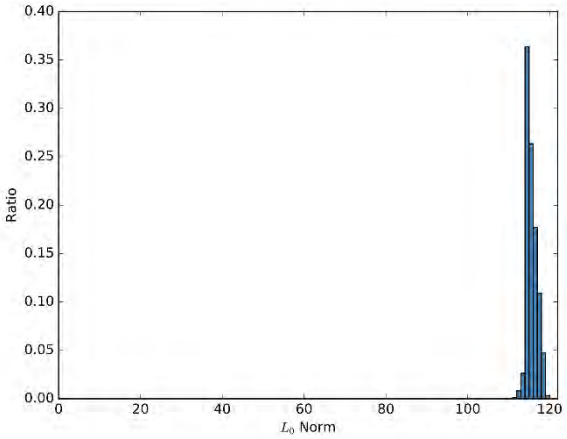

FIGURE 15. L_0 norm for the L_0-norm CW-generated dataset.

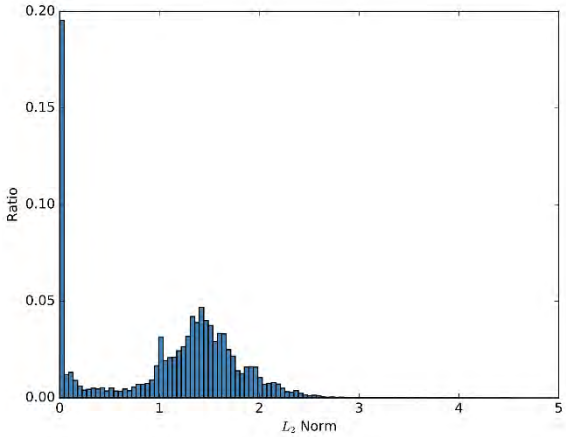

FIGURE 17. L_2 norm for the L_2-norm CW-generated dataset.

TABLE 13. Metrics for the MLP classifier on the L_2-norm CW-generated dataset.

Label	Accuracy	Precision	Recall	False Alarm	F-score
Normal	63.47	54.94	84.38	52.34	66.56
DOS	69.81	60.47	25.33	8.19	35.72
Probe	82.38	26.36	35.91	12.06	30.41
R2L	86.62	44.12	17.16	3.19	24.70
U2R	99.38	9.09	11.94	0.36	10.32

TABLE 14. Metrics for the MLP classifier on the L_∞-norm CW-generated dataset.

Label	Accuracy	Precision	Recall	False Alarm	F-score
Normal	65.06	56.41	82.81	48.41	67.12
DOS	69.56	58.53	27.31	9.57	37.25
Probe	81.31	25.72	39.16	13.60	31.05
R2L	86.75	44.84	16.30	2.94	23.91
U2R	99.38	5.13	5.97	0.33	5.52

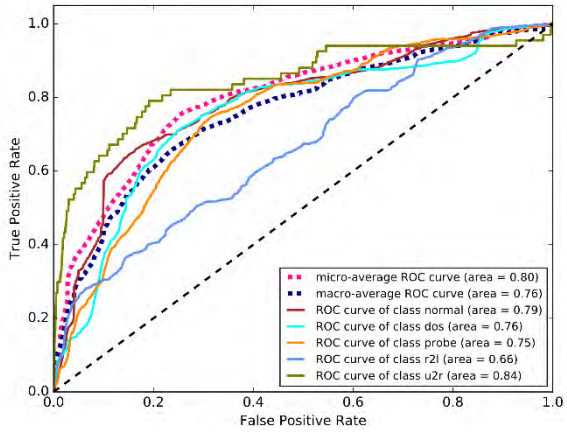

FIGURE 16. ROC curves for the MLP classifier on the L_2-norm CW-generated dataset.

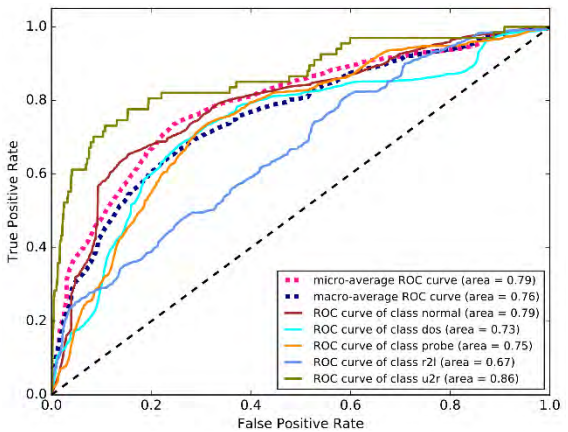

FIGURE 18. ROC curves for the MLP classifier on the L_∞-norm CW-generated dataset.

attacks In Table 15. In Table 15, the upper right cells which are colored in yellow are the intersection of the corresponding two attacks, the lower left cells which are colored in blue are the intersection of the three attacks excluding the corresponding two attacks, and the triangle cells which are colored in green are the intersection of the four attacks excluding the corresponding attacks. The commonly used features for all the five attacks include "dst_bytes", "dst_host_same_src_port_rate", "dst_host_srv_count", "src_bytes", "srv_count", "dst_host_rerror_rate", and "dst_host_same_srv_rate". The intersections in some way indicate the similarities among different attacks. And the 7 common features imply the intrinsic properties of any attacks targeting deep neural networks. That is, the features can be considered as the major contributors when attackers attempt to generate adversarial samples.

Among the commonly used features, we discuss how an adversary can manipulate any of them. "dst_bytes" and "src_bytes" represent the traffic rate between the source and

TABLE 15. Intersection matrix of top 60 features altered for different attacks. The upper right cells which are colored in yellow are the intersection of the corresponding two attacks, the lower left cells which are colored in blue are the intersection of the three attacks excluding the corresponding two attacks, and the triangle cells which are colored in green are the intersection of the four attacks excluding the corresponding attacks.

	JSMA	Untargeted FGSM	Least-Likely FGSM	Random-Target FGSM	DeepFool
JSMA	service_link, flag_SF, dst_bytes, dst_host_same_src_port_rate, service_pop_2, dst_host_srv_count, count, service_echo, service_login, src_bytes, srv_count, service_name, service_ctf, dst_host_rerror_rate, dst_host_same_srv_rate	service_nntp, flag_RSTR, srv_diff_host_rate, service_domain_u, same_srv_rate, dst_host_same_src_port_rate, dst_bytes, dst_host_srv_count, dst_host_count, count, service_urh_i, service_pm_dump, service_gopher, src_bytes, srv_count, service_remote_job, service_pop_3, service_ntp_u, dst_host_rerror_rate, dst_host_same_srv_rate	service_private, dst_host_count, serror_rate, flag_RSTOS0, service_remote_job, service_ftp, urgent, diff_srv_rate, dst_host_same_srv_rate, logged_in, service_imap4, srv_diff_host_rate, same_srv_rate, dst_host_same_src_port_rate, dst_host_srv_count, count, flag_SH, dst_host_srv_rerror_rate, duration, flag_OTH, dst_host_srv_diff_host_rate, num_failed_logins, rerror_rate, hot, service_gopher, src_bytes, srv_count, wrong_fragment, dst_host_rerror_rate, service_eco_i, dst_bytes, dst_host_diff_srv_rate, root_shell, num_compromised, service_pm_dump	service_private, dst_host_count, flag_S3, flag_RSTOS0, service_remote_job, urgent, diff_srv_rate, flag_RSTO, logged_in, service_nntp, srv_diff_host_rate, service_imap4, flag_REJ, dst_host_same_srv_rate, same_srv_rate, dst_host_same_src_port_rate, dst_host_srv_count, count, flag_SH, dst_host_srv_rerror_rate, duration, flag_RSTR, flag_OTH, dst_host_srv_diff_host_rate, num_failed_logins, num_shells, rerror_rate, src_bytes, srv_count, dst_host_rerror_rate, dst_bytes, dst_host_diff_srv_rate, num_access_files, service_ftp	service_private, service_telnet, dst_host_srv_serror_rate, serror_rate, flag_RSTOS0, service_ftp, urgent, diff_srv_rate, dst_host_same_srv_rate, logged_in, service_nntp, service_imap4, dst_host_same_src_port_rate, dst_host_serror_rate, dst_host_srv_count, count, flag_SH, dst_host_srv_rerror_rate, duration, flag_RSTR, dst_host_srv_diff_host_rate, num_failed_logins, rerror_rate, hot, service_gopher, src_bytes, srv_count, wrong_fragment, dst_host_rerror_rate, service_eco_i, dst_bytes, dst_host_diff_srv_rate, root_shell, is_guest_login, num_compromised, service_pm_dump
Untargeted FGSM	service_private, service_echo, flag_RSTOS0, service_name, urgent, diff_srv_rate, dst_host_same_srv_rate, logged_in, service_imap4, service_link, service_http, dst_host_same_src_port_rate, dst_host_srv_count, count, flag_SH, dst_host_srv_rerror_rate, duration, dst_host_srv_diff_host_rate, num_failed_logins, service_pop_2, rerror_rate, service_login, src_bytes, srv_serror_rate, srv_count, service_ctf, dst_host_rerror_rate, flag_SF, dst_bytes, dst_host_diff_srv_rate, service_ftp	service_private, flag_RSTOS0, urgent, diff_srv_rate, dst_host_same_srv_rate, logged_in, service_imap4, dst_host_same_src_port_rate, count, dst_host_srv_count, flag_SH, dst_host_srv_rerror_rate, duration, dst_host_srv_diff_host_rate, num_failed_logins, rerror_rate, src_bytes, srv_count, dst_host_rerror_rate, dst_bytes, dst_host_diff_srv_rate, service_ftp	service_supdup, dst_host_count, service_echo, srv_rerror_rate, service_name, service_remote_job, service_vmnet, service_aol, dst_host_same_srv_rate, srv_diff_host_rate, service_link, same_srv_rate, dst_host_same_src_port_rate, dst_host_srv_count, count, service_netbios_ns, service_Z39_50, service_pop_2, service_rje, service_login, src_bytes, service_gopher, srv_count, service_discard, service_ctf, dst_host_rerror_rate, service_iso_tsap, service_klogin, flag_SF, dst_bytes, service_hostnames, service_pm_dump	service_supdup, service_time, dst_host_count, service_echo, service_name, service_vmnet, service_remote_job, service_aol, dst_host_same_srv_rate, service_nntp, srv_diff_host_rate, service_link, same_srv_rate, dst_host_same_src_port_rate, dst_host_srv_count, count, flag_RSTR, service_Z39_50, service_whois, service_uucp, service_pop_2, service_netbios_ssn, service_login, src_bytes, srv_count, service_discard, service_ctf, dst_host_rerror_rate, service_netstat, service_iso_tsap, service_klogin, flag_SF, dst_bytes, is_host_login, service_hostnames	service_echo, srv_rerror_rate, service_name, dst_host_same_srv_rate, service_nntp, service_link, dst_host_same_src_port_rate, dst_host_srv_count, count, service_netbios_ns, flag_RSTR, service_whois, service_pop_2, service_uucp, service_rje, service_mtp, src_bytes, service_gopher, srv_count, service_daytime, service_login, service_ctf, service_printer, dst_host_rerror_rate, flag_SF, dst_bytes, service_http_2784, service_pm_dump

the destination. An adversary needs to suppress its traffic rate in a limited degree in order to fool the deep neural networks. ''srv_count'' represents the number of connections to the same service. It implies that a slightly limited count of connections is required for an adversary to generate the misclassification results. Similarly, ''dst_host_srv_count'' which means the host based count of connections to the same service is also subject to minor change in the adversarial attempts. ''dst_host_same_src_port_rate'' and ''dst_host_same_srv_rate'' give the rate of connections corresponding to the same service and port and the same service respectively. Therefore an adversary needs to slightly tweak the distribution of connections among services and ports for a better success rate of misclassification. ''dst_host_rerror_rate'' manifests the host based error rate of connections. It means an adversary should modestly constrain its error connections for a better chance of misclassification.

TABLE 15. *(Continued.)* **Intersection matrix of top 60 features altered for different attacks. The upper right cells which are colored in yellow are the intersection of the corresponding two attacks, the lower left cells which are colored in blue are the intersection of the three attacks excluding the corresponding two attacks, and the triangle cells which are colored in green are the intersection of the four attacks excluding the corresponding attacks.**

	JSMA	Untargeted FGSM	Least-Likely FGSM	Random-Target FGSM	DeepFool
Least-Likely FGSM	service_nntp, flag_RSTR, service_link, flag_SF, service_whois, dst_host_same_src_port_rate, service_pop_2, service_uucp, dst_bytes, dst_host_srv_count, count, service_echo, service_login, src_bytes, srv_count, service_name, service_ctf, dst_host_rerror_rate, dst_host_same_srv_rate	service_private, flag_RSTOS0, urgent, diff_srv_rate, dst_host_same_srv_rate, logged_in, service_nntp, service_imap4, dst_host_same_src_port_rate, count, dst_host_srv_count, flag_SH, dst_host_srv_rerror_rate, duration, flag_RSTR, dst_host_srv_diff_host_rate, num_failed_logins, rerror_rate, src_bytes, srv_count, dst_host_rerror_rate, dst_bytes, dst_host_diff_srv_rate, service_ftp	service_nntp, flag_RSTR, dst_bytes, dst_host_same_src_port_rate, count, dst_host_srv_count, src_bytes, srv_count, dst_host_rerror_rate, dst_host_same_srv_rate	service_private, service_supdup, dst_host_count, service_echo, service_name, service_vmnet, flag_RSTOS0, service_remote_job, urgent, protocol_udp, service_aol, diff_srv_rate, logged_in, dst_host_same_srv_rate, srv_diff_host_rate, service_link, service_imap4, same_srv_rate, dst_host_same_src_port_rate, service_http, dst_host_srv_count, count, flag_SH, dst_host_srv_rerror_rate, duration, service_kshell, flag_OTH, dst_host_srv_diff_host_rate, service_Z39_50, num_failed_logins, service_pop_2, rerror_rate, service_login, src_bytes, srv_serror_rate, srv_count, service_discard, service_ctf, dst_host_rerror_rate, service_iso_tsap, service_klogin, flag_SF, dst_bytes, dst_host_diff_srv_rate, service_hostnames, service_ftp	service_private, service_echo, serror_rate, srv_rerror_rate, service_name, flag_RSTOS0, service_ftp, urgent, diff_srv_rate, dst_host_same_srv_rate, logged_in, service_imap4, service_link, service_http, dst_host_same_src_port_rate, dst_host_srv_count, count, flag_SH, dst_host_srv_rerror_rate, service_netbios_ns, duration, dst_host_srv_diff_host_rate, num_failed_logins, service_pop_2, service_rje, rerror_rate, hot, service_gopher, src_bytes, srv_serror_rate, srv_count, service_login, wrong_fragment, service_ctf, dst_host_rerror_rate, service_eco_i, flag_SF, dst_bytes, dst_host_diff_srv_rate, root_shell, service_harvest, land, num_compromised, service_pm_dump
Random-Target FGSM	service_link, flag_SF, dst_bytes, dst_host_same_src_port_rate, service_netbios_ns, service_pop_2, service_rje, dst_host_srv_count, count, service_pm_dump, service_echo, service_gopher, src_bytes, service_login, srv_count, srv_rerror_rate, service_name, service_ctf, dst_host_rerror_rate, dst_host_same_srv_rate	service_private, serror_rate, flag_RSTOS0, service_ftp, urgent, diff_srv_rate, dst_host_same_srv_rate, logged_in, service_imap4, dst_host_same_src_port_rate, count, dst_host_srv_count, flag_SH, dst_host_srv_rerror_rate, duration, dst_host_srv_diff_host_rate, num_failed_logins, hot, rerror_rate, service_gopher, src_bytes, srv_count, wrong_fragment, dst_host_rerror_rate, service_eco_i, dst_bytes, dst_host_diff_srv_rate, root_shell, num_compromised, service_pm_dump	service_nntp, dst_bytes, dst_host_same_src_port_rate, dst_host_same_srv_rate, count, dst_host_srv_count, service_gopher, src_bytes, srv_count, service_pm_dump, dst_host_rerror_rate, flag_RSTR	dst_bytes, dst_host_same_src_port_rate, dst_host_srv_count, count, service_gopher, src_bytes, srv_count, service_pm_dump, dst_host_rerror_rate, dst_host_same_srv_rate	service_private, service_echo, flag_RSTOS0, service_name, urgent, diff_srv_rate, dst_host_same_srv_rate, logged_in, service_nntp, service_imap4, service_link, service_http, dst_host_same_src_port_rate, dst_host_srv_count, count, flag_SH, dst_host_srv_rerror_rate, duration, flag_RSTR, dst_host_srv_diff_host_rate, num_failed_logins, service_whois, service_pop_2, service_uucp, rerror_rate, service_login, src_bytes, srv_serror_rate, srv_count, service_ctf, dst_host_rerror_rate, flag_SF, dst_bytes, dst_host_diff_srv_rate, service_ftp
DeepFool	service_supdup, dst_host_count, service_echo, service_name, service_vmnet, service_remote_job, service_aol, dst_host_same_srv_rate, srv_diff_host_rate, service_link, same_srv_rate, dst_host_same_src_port_rate, dst_host_srv_count, count, service_Z39_50, service_pop_2, service_login, src_bytes, srv_count, service_discard, service_ctf, dst_host_rerror_rate, service_iso_tsap, service_klogin, flag_SF, dst_bytes, service_hostnames	service_private, dst_host_count, flag_RSTOS0, service_remote_job, urgent, diff_srv_rate, dst_host_same_srv_rate, logged_in, service_imap4, srv_diff_host_rate, same_srv_rate, dst_host_same_src_port_rate, count, dst_host_srv_count, flag_SH, dst_host_srv_rerror_rate, duration, flag_OTH, dst_host_srv_diff_host_rate, num_failed_logins, rerror_rate, src_bytes, srv_count, dst_host_rerror_rate, dst_bytes, dst_host_diff_srv_rate, service_ftp	service_nntp, srv_diff_host_rate, same_srv_rate, dst_host_same_src_port_rate, dst_bytes, dst_host_same_srv_rate, count, dst_host_count, dst_host_srv_count, src_bytes, srv_count, service_remote_job, dst_host_rerror_rate, flag_RSTR	srv_diff_host_rate, dst_bytes, dst_host_same_src_port_rate, same_srv_rate, count, dst_host_count, dst_host_srv_count, service_gopher, src_bytes, srv_count, service_remote_job, service_pm_dump, dst_host_rerror_rate, dst_host_same_srv_rate	srv_diff_host_rate, same_srv_rate, dst_host_same_src_port_rate, dst_bytes, count, dst_host_count, dst_host_srv_count, src_bytes, srv_count, service_remote_job, dst_host_rerror_rate, dst_host_same_srv_rate

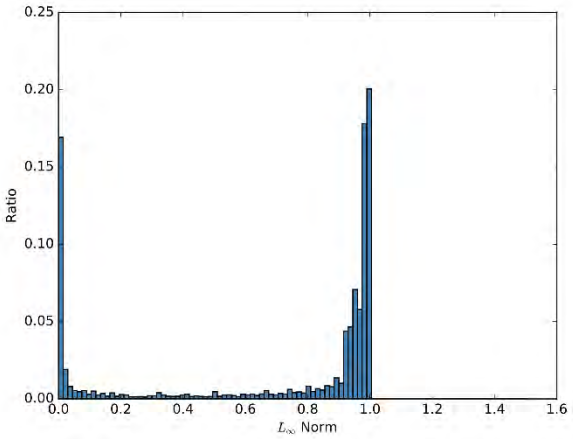

FIGURE 19. L_∞ norm for the L_∞-norm CW-generated dataset.

While we focus on adversarial examples against a constant known model in this study, it is possible that adversarial examples targeting one model will also mislead other models. It was first reported by Szegedy *et al.* [6] that adversarial examples generated based on a neural network can fool the same neural networks trained by different datasets. The property of transferring adversarial examples across different models is called transferability. Papernot *et al.* [9] discovered another sort of transferability which let adversarial examples generated based on a neural network apply to other neural networks with different architectures, even other classifiers trained by different machine learning algorithms. In some way, transferability can be leveraged to turn black-box attack to white-box attacks. In the black-box attack setting, the adversary has no knowledge of the target model (e.g. architecture and parameters) and no access to the training dataset. Nevertheless, the adversary can construct adversarial examples based on a self-made model (which intends to mimic the target model) and train the model using a dataset which has some similarity with the training dataset used by the target model. Then thanks totransferability, those adversarial examples which are crafted targeting the self-made model are also likely to fool the target model. Thus, the white-box attack studied in this paper is also meaningful for a pure black-box attack.

VII. CONCLUSION AND FUTURE WORK

In this paper, we evaluated the state-of-the-art attack algorithms in the deep learning based intrusion detection domain. We found the attack algorithms, which were originally proposed to fool the deep learning based image classifier, demonstrated different levels of effectiveness in the intrusion detection domain. We identified the different feature usage patterns for the attack algorithms. In practice, an adversary has limited resources and capability to manipulate features. So altering a large set of features is less practical for an adversary in most cases. As JSMA attacks tend to heavily use a limited set of features, they are relatively more attractive for an adversary in terms of usability and applicability. We also noted the varying degrees of significance across features in terms of their rates of being selected to be perturbed by an adversary. The most commonly used features indicate they contribute more to the vulnerability of the deep learning based intrusion detection and therefore they deserve more attention and better protection in the detection and defense efforts.

As transferability is often fundamentally critical for black-box attacks, the future work will focus on the transferability of adversarial examples in deep learning based intrusion detection. It will be studied in three dimensions: 1) transferability within the same neural network trained with different inputs; 2) transferability among different neural networks; 3) transferability between well-known conventional machine learning algorithms (i.e., random forest, SVM, decision tree) and deep neural networks.

REFERENCES

[1] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 1, pp. 41–50, Feb. 2018.
[2] 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. 21954–21961, 2017.
[3] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proc. 9th EAI Int. Conf. Bio-Inspired Inf. Commun. Technol. (BICT)*, 2015, pp. 21–26.
[4] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. Int. Conf. Wireless Netw. Mobile Commun. (WINCOM)*, Oct. 2016, pp. 258–263.
[5] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.
[6] C. Szegedy *et al.* (2013). "Intriguing properties of neural networks." [Online]. Available: https://arxiv.org/abs/1312.6199
[7] I. J. Goodfellow, J. Shlens, and C. Szegedy. (2014). "Explaining and harnessing adversarial examples." [Online]. Available: https://arxiv.org/abs/1412.6572
[8] A. Kurakin, I. Goodfellow, and S. Bengio. (2016). "Adversarial examples in the physical world." [Online]. Available: https://arxiv.org/abs/1607.02533
[9] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. IEEE Eur. Symp. Secur. Privacy*, Nov. 2015, pp. 372–387.
[10] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A simple and accurate method to fool deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2574–2582.
[11] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Secur. Privacy*, Mar. 2017, pp. 39–57.
[12] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl. (CISDA)*, Jul. 2009, pp. 1–6.
[13] *KDD Cup 99 Dataset*. Accessed: Apr. 21, 2018. [Online]. Available: http://kdd.ics.uci.edu/ databases/kddcup99/kddcup99.html
[14] *NSL-KDD Dataset*. Accessed: Apr. 21, 2018. [Online]. Available: http://www.unb.ca/cic/research/datasets/nsl.html
[15] *TensorFlow*. Accessed: Apr. 21, 2018. [Online]. Available: https://www.tensorflow.org

ZHENG WANG received the Ph.D. degree in computer science from the Computer Network Information Center, Chinese Academy of Sciences, in 2010. He is currently a Research Associate with the Information Technology Laboratory, National Institute of Standards and Technology, USA. He is a Co-Inventor of six patents and had co-authored over 30 academic papers. His research interests include security and privacy, machine learning, network measurement, and Internet naming and addressing.