SPECIAL SECTION ON ARTIFICIAL INTELLIGENCE IN PARALLEL AND DISTRIBUTED COMPUTING

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 27, 2019, accepted December 23, 2019, date of publication January 9, 2020, date of current version January 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2965184

Adversarial Examples Detection for XSS Attacks Based on Generative Adversarial Networks

XUEQIN ZHANG[1], YUE ZHOU[1], SONGWEN PEI[2], (Senior Member, IEEE), JINGJING ZHUGE[1], AND JIAHAO CHEN[1]

[1]Department of Electronic and Communications Engineering, East China University of Science and Technology, Shanghai 200237, China
[2]Department of Computer Science and Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China

Corresponding authors: Xueqin Zhang (zxq@ecust.edu.cn) and Songwen Pei (swpei@usst.edu.cn)

This work was supported by the Natural Science Foundation of China under Grant NNSFC 61472139.

ABSTRACT Models based on deep learning are prone to misjudging the results when faced with adversarial examples. In this paper, we propose an MCTS-T algorithm for generating adversarial examples of cross-site scripting (XSS) attacks based on Monte Carlo tree search (MCTS) algorithm. The MCTS algorithm enables the generation model to provide a reward value that reflects the probability of generative examples bypassing the detector. To guarantee the antagonism and feasibility of the generative adversarial examples, the bypassing rules are restricted. The experimental results indicate that the missed detection rate of adversarial examples is significantly improved after the MCTS-T generation algorithm. Additionally, we construct a generative adversarial network (GAN) to optimize the detector and improve the detection rate when dealing with adversarial examples. After several epochs of adversarial training, the accuracy of detecting adversarial examples is significantly improved.

INDEX TERMS Network intrusion detection, generative adversarial network, Monte Carlo tree, convolutional neural networks.

I. INTRODUCTION

Deep learning methods, with their high-precision in classification and high-speed processing performance, are expected to complement or replace traditional intrusion detection technologies in the detection of intrusions under complex internet environments. Since being proposed by Hinton and Salakhutdinov [1], deep learning has proven to have excellent performance in fields such as image classification [2], [3] and data mining [4].

However, deep learning technology tends to perform poorly when faced with adversarial examples. Adversarial examples can trick machine learning models with minor adjustments of the original samples, making machine learning models produce incorrect outputs [5]. Adversarial examples have been proven to have strong misleading effects in the field of image recognition [6]. Therefore, adversarial examples have become an unavoidable problem when developers apply deep learning models to practical issues, especially in the information security field. Defending against attacks caused by adversarial examples and minimizing the effectiveness of adversarial examples are important tasks for deep learning experts in the security field.

Adversarial examples first appeared in the image classification field [8]. Adversarial examples take advantage of the highly nonlinear characteristic of the neural network to deceive the model through minor changes to the original samples, making machine learning models produce incorrect classification decisions.

Assuming that function g is defined in sample space X and that $g(x){>}0$ denotes that the classification result is true, the attacker's goal is to design a sample x^* that makes $g(x^*)\ >0$. Therefore, the following optimal object can be defined as (1):

$$x^* = \arg\max_x \hat{g}(x), \quad s.t.\ d(x, x^*) \le d_{\max} \tag{1}$$

Restrictions specify that the normal distance between the generated image and the original image must be no larger than a certain threshold, which means there can only be slight visual disturbances to the picture. The image can be misclassified with high confidence after this minor change that is hard to notice with human eyes. However, unlike image data, for XSS attack traffic data, a small local change to the original data can ruin the function of the traffic, so the changes can

The associate editor coordinating the review of this manuscript and approving it for publication was Tao Li.

be done to XSS traffic are restricted in this paper. Since the bypassing rules of the generation model are limited, we use a probabilistic simulation method rather than a gradient descent method to ensure feasibility; thus, we use a Monte Carlo tree search (MCTS) algorithm to generate adversarial traffic examples.

Adversarial training is a training method that can be used to defend against adversarial examples. The idea is adding adversarial examples into datasets at the model training stage, using a mixture of adversarial examples and original samples as the training sets. Adversarial training can be integrated into a GAN model to dynamically generate adversarial examples. The GAN-based adversarial training method is an advancement of deep learning technology. By alternately training the generator and the discriminator, the discrimination model can detect an increasing number of generated adversarial examples and update itself to defend against them. Since the search scope of the generator is limited by bypassing rules, the discriminator can theoretically cover all adversarial situations. Using GAN, the discrimination model can avoid traversing all attack samples and accelerate convergence when defending against adversarial examples.

In this paper, we proposed an improved MCTS algorithm to generate adversarial examples of XSS attack traffic data. The proposed MCTS algorithm guarantees the feasibility of the generated adversarial traffic examples and provides a reward for assessing the performance of each bypassing operation. Additionally, we built a GAN network to optimize the intrusion detection model to defend against adversarial attacks. The experimental results showed that our method can effectively obtain adversarial examples, and the GAN-optimized detection model can effectively defend against such adversarial examples.

II. RELATED WORK

Deep learning technology has been used in the field of information security. For intrusion detection, Nathan et al. proposed a nonsymmetric deep autoencoder (NDAE) based on a deep autoencoder (DAE). The proposed model cascades two NDAEs at the end of the network, and uses random forest as the classifier. It achieved an average accuracy of 97.85% on the KDD datasets [7]. Vartouni *et al.* studied HTTP traffic. They used an n-gram model to construct a feature vector from the original internet traffic using isolation forest (iForest) for classification [8]. The above experiments show that the application of deep learning techniques in intrusion detection can lead to high precision and great performance.

Regarding models based on deep learning, adversarial examples are a serious threat that can be exploited by hackers. Grosse Kathrin *et al.* [9] proposed an adversarial examples generation algorithm for malware detection, which allowed 63% of malware to successfully bypass detectors constructed by neural networks. Besides, Tang *et al.* [10] ASG algorithm for Android malware also bypasses the CNN-based Android malware detector without affecting the original malware functions. However, regarding the input data form of the network model, the above two algorithms both extract the features of the original malware samples and construct specific binary vectors to indicate whether a software contains a series of behaviors or introduces a specific library file. This method is ineffective for byte stream-oriented detection models or other feature extraction methods; therefore, its application scope is limited. Al-Dujaili [11] adopted a saddle point optimization method and added adversarial examples to the training of a detection network to improve the detection rate of adversarial examples and the robustness of the detection model. However, this method also extracts the feature vectors obtained from feature extraction, rather than using the original byte stream; therefore, the model is very vulnerable to the influence of the feature extraction algorithm. Such a method is also used in [12]. Kreuk Felix *et al.* used two methods called mid-file injection and end-of-file injection to restrict the adversarial examples to changing only the useless parts of malicious software. Using the fast gradient sign method (FGSM) algorithm, adversarial examples were generated under the premise that the function of malicious software was unchanged, which also effectively deceived the original detector. However, upgrading the detection model to cope with adversarial examples is not discussed. Compared with the aforementioned methods, we proposed an improved MCTS algorithm for generating adversarial examples that guarantees both antagonism and feasibility and is more effective than FGSM.

The GAN-based adversarial training method is a useful way to improve the discrimination model to defend against adversarial examples. Jie Cao et al. used a GAN to build a pose-invariant model for human face recognition, which has effectively improved the recognition rate of the model [13]. Researchers at Carnegie Mellon University used a GAN to improve the target detection rates of occluded objects [14]. However, few studies have applied GANs to intrusion detection technologies.

In this paper, we proposed a GAN-based adversarial training method named MCTS-T to defend against adversarial examples. To ensure the effectiveness of generated XSS adversarial samples, two factors of ‘‘modification position’’ and ‘‘modification action’’ are considered synthetically in the MCTS-T algorithm. Compared to the traditional Monte Carlo tree algorithm, the improvements in MCTS-T are as follows: 1)In the selection stage, an improved UCB method called UCB-T is proposed to select the best action sequence and the corresponding adversarial examples for XSS attacks. 2)In the expansion stage, a dropout strategy is applied to increase the generalization ability of the MCTS-T model. 3)In the simulation stage, feedback is obtained by the detection model proposed in Section 2.2.

III. GENERATING XSS ADVERSARIAL EXAMPLES

The XSS attack is one of the most common vulnerabilities in web applications. It induces users to execute malicious scripts by inserting HTML or JavaScript code into the effective input area of a website stealing user information. Deep learning models have proved to be useful for defending against normal XSS attacks. However, few models are designed for adversarial examples. In this section, we proposed an MCTS-T algorithm to generate adversarial examples of XSS attacks. The proposed MCTS-T generation algorithm is specifically designed as a black model, which means that the attackers do not need to know the internal configurations of the detectors to bypass an intrusion detector. Therefore, our generation algorithm can be easily applied in practice.

A. XSS BYPASSING RULE SETS

Since we need to ensure the feasibility of our generated attacks, we cannot modify the original data at will. In this section, we summarized the following bypassing rules based on OpenAI Gym for XSS attacks:

- Hexadecimal Encoding: After hexadecimal encoding, tag content can still be parsed by an escape browser. For example, the hexadecimal escape of letter s is "&#0x73" or "&#x73".
- Decimal Encoding: Similar to hexadecimal encoding, decimal encoding is available for bypassing. For example, the decimal escape of letter s is "&#115".
- URL Encoding: URL encoding is used by browsers for packaging from the input. It can take the form of "% + hexadecimal encoding". For example, the corresponding URL encoding of letter s is %73. In addition, unlike hexadecimal encoding, URL encoding can be applied to any place on the submitting form, rather than being limited to the tag.
- Inserting Invalid Characters: Inserting invalid characters such as space, "/0", or enter in the middle of the tag content will not affect the browser's normal parsing but can cut off keywords.
- Case Mixture: HTML tags are not case sensitive. Changing the content case of the HTML tag will not affect the normal parsing of browsers.

The proposed MCTS-T generation algorithm can modify only the original XSS attacks according to the aforementioned rules to ensure that the modified attacks are still valid.

B. XSS DETECTION MODE

To verify the validity of our algorithm for deep learning detection models, we build a deep learning anomaly detection model by referring to paper [15]. The model first converts hexadecimal traffic data into decimal data one by one. Because each data point is represented by double hexadecimal digits in the traffic, the maximum value of each data point is 255 in decimal and the range of the image pixel value is 0–255. During the preprocessing stage, the model converts the original traffic data flow into a two-dimensional image structure. CNN is used to learn the spatial features of the traffic image. The multilayer CNN structure extracts high-dimensional features from the original traffic image, outputting them as a one-dimensional vector.

The model adopts the default structure and hyperparameters of Alexnet, which contains eight hidden layers. Each of the first six hidden layers consists of a CNN layer and a pooling layer. The last two hidden layers are full connection layers that integrate the CNN outputs and send them to a SoftMax layer. Finally, the SoftMax layer predicts the result according to the feature vector and propagates loss backward by updating the parameters of CNN.

The model is trained only by the original XSS attacks dataset. When the training of the model has completely converged, and its loss and accuracy no longer have obvious changes, we use the fully convergent detection model as our baseline to verify whether our generation algorithm can bypass it. Note that we also compared in-depth learning models; details can be found in the experimental section.

C. GENERATING ADVERSARIAL EXAMPLES BASED ON MCTS-T

According to reinforcement learning theory, the variation operation of XSS traffic is called an action, and the traffic samples before and after the action are referred to as different states. As mentioned in Section 3.1, each bypass rule can be defined as an action. Therefore, generating adversarial examples involves determining the action sequence. We proposed an MCTS-based algorithm to find the optimum action sequence and generate the corresponding adversarial examples.

MCTS is a common search algorithm in computer science. It constructs an incomplete search tree and estimates a globally optimal solution according to the simulation results of the partial state sequence. Therefore, the MCTS-based algorithm can provide a reward value for a black-box model. There are four main steps in the MCTS algorithm:

Selection: Start traversing from the root node to a deeper node until reaching a subsequent state node that has not been visited.

Expansion: Add a subsequent state node to the tree.

Simulation: Start simulating from the new state, i.e., perform random operations according to the action set before reaching the maximum depth.

Feedback: The loss of the final state propagates backward, updating parameters of the selected nodes in the tree

Different from the traditional MCTS algorithms, we proposed an MCTS-T algorithm specifically designed for XSS attacks and applied to generate adversarial examples.

Assuming that the current state of the sequence is S_i (S_0 presents the original state of the sequence), there are two factors determining the next state S_{i+1}, one is the modification position and the other is the action corresponding to each position. In the selection stage, we use the Upper Confidence Bounds (UCB) algorithm to choose the best path. The UCB algorithm is a classical selection algorithm often used by MCTS. The advantage is that it can take into account the breadth and depth of search, and has a good global optimization ability. For each node, a UCB value determines whether we choose the current node. The traditional UCB is

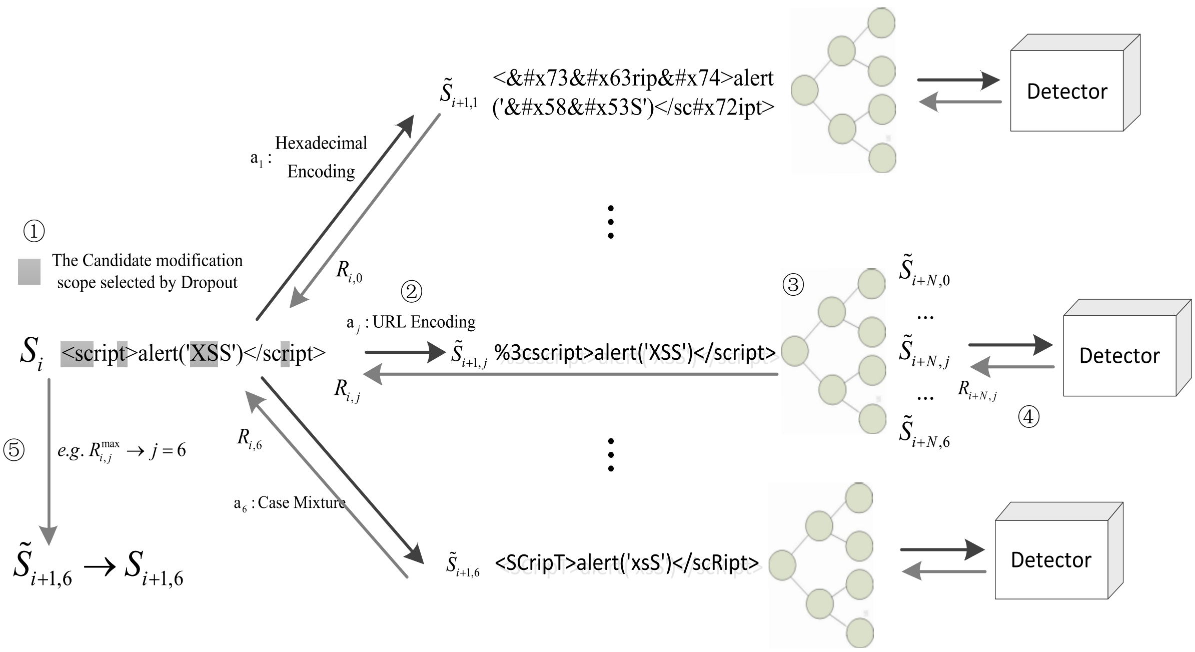

$\tilde{S}_{i+1,1}$ <&#x73&#x63rip&#x74>alert ('&#x58&#x53S')</sc#x72ipt>
a_1 : Hexadecimal Encoding
Detector
① The Candidate modification scope selected by Dropout
$R_{i,0}$
②
a_j : URL Encoding
S_i <script>alert('XSS')</script>
$\tilde{S}_{i+1,j}$ %3cscript>alert('XSS')</script>
$R_{i,j}$
③
$\tilde{S}_{i+N,0}$
...
$\tilde{S}_{i+N,j}$
...
$\tilde{S}_{i+N,6}$
Detector
$R_{i+N,j}$
④
⑤ $e.g.\ R^{\max}_{i,j} \to j=6$
$R_{i,6}$
a_6 : Case Mixture
$\tilde{S}_{i+1,6}$ <SCripT>alert('xsS')</scRipt>
Detector
$\tilde{S}_{i+1,6} \to S_{i+1,6}$

FIGURE 1. Process of generating an adversarial example of XSS attacks using the MCTS-T algorithm.

defined as (2):

$$UCB = \frac{R_j}{N_j} + \sqrt{\frac{C \times lnN}{N_j}} \tag{2}$$

Where R_j represents the total reward of the j^{th} child node, N_j is the number of times the j^{th} child node is selected. N is the number of times the current node is selected. So the first part of this formula is the utilization of existing knowledge, and the second part is the exploration of inadequately simulated nodes. Parameter C controls the balance of two parts. However, such UCB formula is not suitable for XSS attacks. For the issue of verifying XSS attacks, we have two decision part. First is to choose the candidate modification scope; Second is to choose the bypassing action. Besides, we record the number of times each location has been selected. We define the reward of position k as R_pos_k and the times as N_pos_k. For each bypassing action, we also define the reward and times of action j as R_act_j and N_act_j. Therefore, the improved *UCB-T* algorithm for MCTS-T is a Cartesian product of the candidate modification position and the bypassing action for each position.

$$UCB-T = \frac{R_{pos_k} \times R_{act_j}}{N_{pos_k} \times N_{act_j}} + \sqrt{\frac{C_1 \times lnN_{pos}}{N_{pos_k}}} + \sqrt{\frac{C_2 \times lnN_{act}}{N_{act_j}}} \tag{3}$$

where C_1 and C_2 represent the factor of choosing a new position and a new act. When we find the position pos_k and action act_j corresponding to the maximum *UCB-T* value, the next state S_{i+1} can be determined.

In the expansion stage, a dropout method is used to choose the candidate modification position randomly. At the beginning of the stage, for each single data in the traffic, we stochastically ignore it with a certain probability p ($p = 0.5$ usually). The ignored data will not be processed this time, which makes the generative samples more generalized. The words with gray backgrounds shown in Fig. 1 are the examples of the candidate modifications generated by the dropout method and will change randomly in each selection stage. All actions in the current state are traversed, and for each action a_j, the sequence state $\tilde{S}_{i+T,j}$ obtained by the action is saved. We repeat the above action N times and obtain the final state $\tilde{S}_{i+N,j}$.

In the simulation stage, for each final state $\tilde{S}_{i+N,j}$ of the sequence, it will be sent into the XSS detection model D described in Section 2.2 and obtain the SoftMax output as confidence $\Delta R_{i+N,j}$, which represents the traffic sequence classified as normal or attack.

$$\Delta R_{i+N,j} = SoftMax(D(\tilde{S}_{i+N,j})) \tag{4}$$

In the MCTS, for current state S_i, the feedback of action a_j is $\Delta R_{i,j}$. It equals the weighted average value of $\Delta R_{i+1,j}$, that is

$$\Delta R_{i,j} = \frac{\sum_j w_j \cdot R_{i+1,j}}{\sum_j w_j} \tag{5}$$

where w_j represents a predefined weight associated with the probability of each action. It can be seen from (5), $\Delta R_{i,j}$ is obtained by $\Delta R_{i+1,j}$, and the $\Delta R_{i+1,j}$ is obtained by $\Delta R_{i+2,j}$, and so on. The $\Delta R_{i+N,j}$ is obtained by Formula (4). The reward is used as feedback in a sequential layer-based process until the i^{th} layer is attached. After the reward value $\Delta R_{i,j}$ of each action a_j is obtained, we update $R\_act_{i,j}$ and $R\_pos_{i,k}$ in *UCB-T* formula as (6) and (7):

$$R\_act_{i,j} = R\_act_{i,j} + \Delta R_{i,j} \tag{6}$$

$$R\_pos_{i,k} = R\_pos_{i,k} + \Delta R_{i,j} \tag{7}$$

Finally, when the search process is finished, the best sequence is determined according to the *UCB-T* formula, and the corresponding adversarial example can also be determined.

IV. GAN-OPTIMIZED DETECTION MODEL

Deep learning detectors often have difficulty detecting adversarial examples. To address this problem, we add adversarial examples into the discrimination model training set, labeling them as attacks and then retrain the model. The process of generating and training is repeated until the deep learning model can stably detect adversarial examples. After the deep learning network converges, the best structure for input interference resistance is achieved.

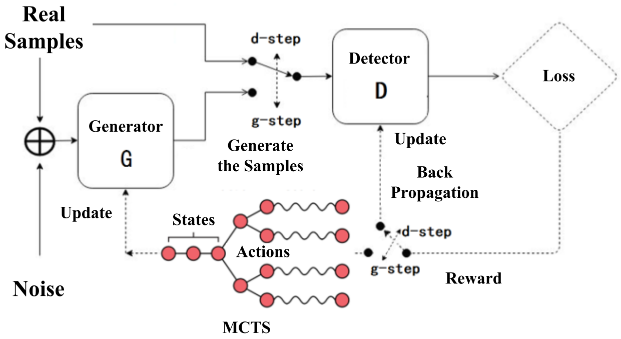


FIGURE 2. Structure of the GAN.

The structure of the GAN adopted by this paper is shown in Fig. 2. The generator G gives a state transition probability decided by Q to maximize its expected end reward:

$$J(\theta) = \sum_{a_j \in A} G_\theta(s_{i+1,j}|s_i) \cdot Q_{D\phi}^{G_\theta}(s_i, a_j) \tag{8}$$

where $Q_{D\phi}^{G_\theta}(s_0, a_j)$ represents the action-value function of a_j obtained by the Monte Carlo search of discrimination net (in Section 2.3). Thus, a generator is more likely to generate adversarial examples that can effectively bypass the detector.

Each round of updating the parameters of the discrimination model is the detection model's learning process of the newly generative adversarial examples. For the discrimination net, the objective loss function is as (9):

$$\begin{aligned}\min_{\phi} -E_{Y\sim p_{normal}}[\log D_\phi(Y)] - E_{Y\sim p_{attack}}[\log(1 - D_\phi(Y))] \\ -E_{Y\sim G_\theta}[\log(1 - D_\phi(Y))]\end{aligned} \tag{9}$$

where $Y \sim P_{normal}$ represents real normal traffic data, $Y \sim P_{attack}$ represents real attack traffic data, and $Y \sim G_\theta$ represents generative XSS adversarial examples.

The generation net and discrimination net carry out minimax two-player games, alternately updating parameters. The specific process is as follows:

Algorithm 1 Training Steps of XSS Traffic Generation Net

```
Require: Generator $G_\theta$; Detector $D_\theta$; Training samples $S_0$;
Randomly initialize parameters $\theta$ of $G_\theta$, $D_\theta$
Pretrain $D_\theta$ with original training samples $S_0$
do
    empty adversarial set $A$
    for each step in g-steps do
        generate XSS adversarial example $s_T$
        for t = 1:T do
            MCTS calculate action-value $Q(s_t, a_j)$
        end
        Calculate loss according to value $Q$ and update
        generation net $G_\theta$ by Eq. (6)
    end
    for each step in d-steps do
        Train net $D_\theta$ with original training samples and
adversarial examples
        update net $D_\theta$ according to the loss by Eq. (7)
    end
while$D_\theta$ converged
```

V. EXPERIMENTAL RESULTS AND ANALYSIS

In this paper, the generation algorithm of adversarial examples of XSS attacks is studied for the first time. Since no similar algorithm has been proposed, we perform ablation experiments to verify the effectiveness of our proposed algorithm. We use the deep learning framework Caffe to build the XSS intrusion detection model. The PC configuration in this experiment is a Core i5-7400 CPU, GTX 1060 graphics card, 8G memory, and Ubuntu 16.04 system.

A. DATASETS AND EVALUATION METRIC

To include the new type of XSS attack, an intrusion detection evaluation dataset called CICIDS2017 [16] is used. This dataset contains benign and up-to-date common attacks based on real-world data (PCAPs). Labeled files based on timestamp, source and destination IPs, source and destination ports, protocol, and attack are provided in this dataset by the CICFlowMeter tool. The damn vulnerable web app (DVWA) is used to automate the attacks in XSS. For this experiment, we extract 4789 XSS attack traffic examples and 12000 normal traffic examples from the dataset. Table 1 shows the quantitative distribution of the training set and the validation set.

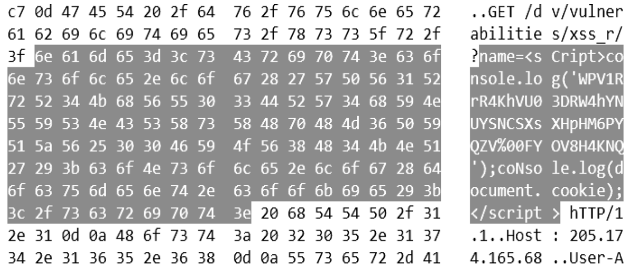
The main evaluating indicators used in the experiment are recall rate (TPR) and precision:

$$TPR = \frac{TP}{TP + FN} \times 100\% \tag{10}$$

$$precision = \frac{TP}{TP + FP} \times 100\% \tag{11}$$

TABLE 1. Quantitative distribution of datasets.

Category	Train	Validation	Total
XSS	3592	1197	4789
Normal	9000	3000	12000
Total	12592	4197	16789



```
c7 0d 47 45 54 20 2f 64  76 2f 76 75 6c 6e 65 72   ..GET /d v/vulner
61 62 69 6c 69 74 69 65  73 2f 78 73 73 5f 72 2f   abilitie s/xss_r/
3f 6e 61 6d 65 3d 3c 73  43 72 69 70 74 3e 63 6f   ?name=<s Cript>co
6e 73 6f 6c 65 2e 6c 6f  67 28 27 57 50 56 31 52   nsole.lo g('WPV1R
72 52 34 4b 68 56 55 30  33 44 52 57 34 68 59 4e   rR4KhVU0 3DRW4hYN
55 59 53 4e 43 53 58 73  58 48 70 48 4d 36 50 59   UYSNCSXs XHpHM6PY
51 5a 56 25 30 30 46 59  4f 56 38 48 34 4b 4e 51   QZV%00FY OV8H4KNQ
27 29 3b 63 6f 4e 73 6f  6c 65 2e 6c 6f 67 28 64   ');coNso le.log(d
6f 63 75 6d 65 6e 74 2e  63 6f 6f 6b 69 65 29 3b   ocument. cookie);
3c 2f 73 63 72 69 70 74  3e 20 68 54 54 50 2f 31   </script > hTTP/1
2e 31 0d 0a 48 6f 73 74  3a 20 32 30 35 2e 31 37   .1..Host : 205.17
34 2e 31 36 35 2e 36 38  0d 0a 55 73 65 72 2d 41   4.165.68 ..User-A
```

FIGURE 3. An example of the XSS attack traffic variant.

where TP denotes the number of samples correctly identified as XSS attacks, FP denotes the number of normal samples incorrectly identified as XSS attacks, and FN denotes the number of XSS samples incorrectly identified as normal traffic.

TPR represents the proportion of correctly identified XSS attacks in all XSS attacks, and it reflects the rate of false negatives, i.e., 1−TPR. Precision represents the proportion of real XSS attacks in all samples identified by the model as XSS, and it reflects the rate of false positives, i.e., 1−precision.

B. EXPERIMENTAL RESULTS

First, we tested the performance of the original XSS attack detection model (i.e., the discrimination net used in GAN). XSS variants of XSS attacks in the validation set were generated through adversarial examples generating the method described in Section 2. To ensure the robustness of the results, ten variants were generated for the original XSS traffic, which means that the ratio of adversarial examples to the original examples was 10:1. Fig. 3 shows an example of an XSS attack traffic variant.

In this experiment, we tested the original XSS detector with the original dataset and the adversarial dataset. We also compared the MCTS-T proposed in this paper with the traditional FGSM [17]. The experimental results are shown in Table 2.

TABLE 2. Detection results with different generation algorithms.

Data Source	*TPR* (%)	*Precision* (%)
Original	96.16	99.91
D_ FGSM	90.14	99.83
D_ MCTS-T	84.32	99.89

Here, D_FGSM and D_MCTS-T means the dataset generated by FGSM and MCTS-T algorithm respectively. The purpose of an algorithm for generating adversarial examples is to confuse the detection models. The higher the false-negative rate (the lower the TPR) is, the better the algorithm. The table shows that the performance of the original detector is degraded by the adversarial examples and that the false-negative rate increases, which proves the effectiveness of the adversarial examples. Additionally, the MCTS-T algorithm is better than the traditional FGSM algorithm, increasing the false-negative rate of the detector by approximately 6%. The precision reflects the false alarm rate of the model. With a precision above 99%, the deep-learning-based XSS detection model has a very low false-positive rate.

To further validate the effectiveness of our generated adversarial dataset, we applied the original dataset and the adversarial dataset to LSTM [18] and XGBoost [19]. The hyperparameters of the comparison classifiers are shown in Table 3. The experimental results are shown in Table 4.

TABLE 3. Structure of comparison classifiers.

classifier	structure and settings
LSTM	Layers: 2 Hidden units: 128 Learning rate: 0.001 Epoch: 100

XGBoost	n_estimators=50
	max_depth=3
	Learning_rate=0.1

TABLE 4. Experimental results with comparison classifier.

Model+Data Source	*TPR* (%)	*Precision* (%)
LSTM+Original	97.74%	98.29%
LSTM+D_MCTS-T	81.23%	99.90%
XGBoost+Original	98.87%	98.87%
XGBoost+D_MCTS-T	39.41%	99.86%

As mentioned above, the decline in *TPR* indicates the effectiveness of adversarial examples. As shown in Table 4, when

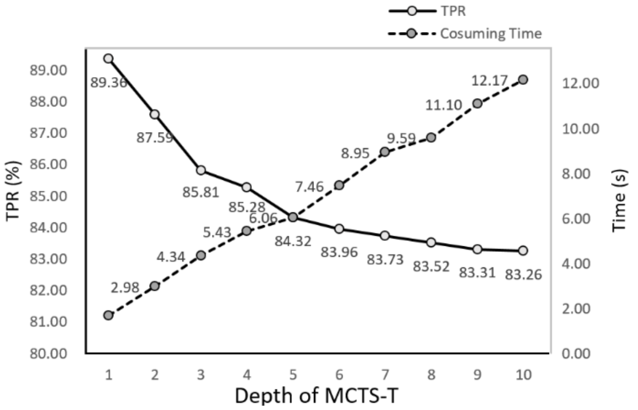


FIGURE 4. Effects of MCTS-T depth on TPR and time consumption.

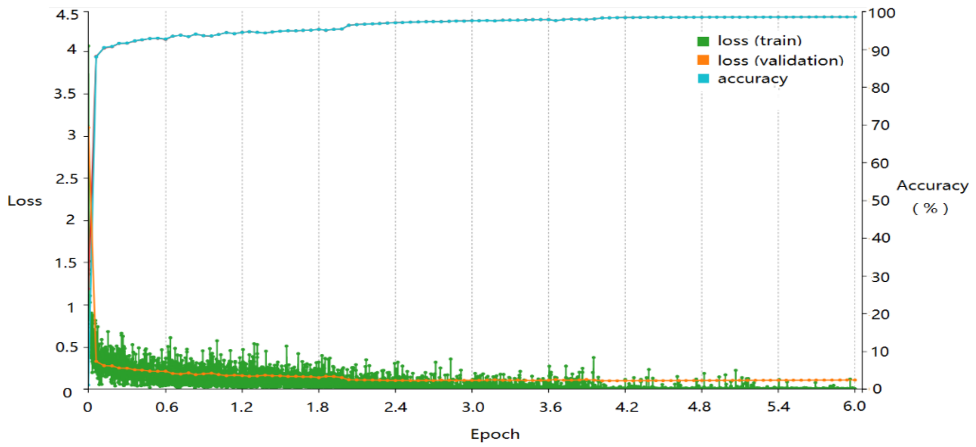


FIGURE 5. Convergence curve of the discrimination model in GAN. The blue curve shows the accuracy of the validation sets. The green and orange curves show the loss of the training set and the validation set, respectively.

TABLE 5. Effects of MCTS-T depth.

Depth of MCTS-T	*TPR* (%)	Time (s)
T=1	89.36	1.676
T=3	85.81	4.337
T=5	84.32	6.055
T=10	82.16	12.173

using adversarial examples, the *TPR* of LSTM decreased by 16.51%, and the *TPR* of XGBoost decreases significantly by 59.46%. The experimental results prove the effectiveness of our adversarial examples. The slight increase in *Precision* is because the number of adversarial examples is higher than the number of original examples, leading to a larger *TP* value in formula (11).

Second, we compare the effect of the MCTS-T's search depth on the search time and effectiveness of adversarial examples. As shown in Table 5 and Fig. 4, with the increase in the search depth of MCTS-T, the detection rate of the detector for adversarial examples decreases significantly. However, an increase in tree depth also leads to an increase in the search time. To balance the effectiveness and generation time of adversarial examples, a search depth of $T = 5$ is used.

We observed the iterative process of GAN to find the change in the detection rate (TPR) for adversarial examples under different numbers of iterations. The results are shown in Table 6 and Fig. 5. For the detection model, the goal of optimization is to defend against adversarial examples; the lower the false-negative rate (the higher the TPR) is, the better the detection model. Table 4 shows that, with the increase in the number of iterations, TPR decreases first and then increases, while precision remains stable. After more than 5 iterations, the TPR and precision remain stable, and the model converges. Therefore, 5 iterations are performed. The experimental results show that the precision remains stable, indicating that the effectiveness of the detection model is not affected during the optimization of GAN.

TABLE 6. Detection rate of adversarial examples with different GAN iterative epochs.

Iterative Epoch	*TPR* (%)	*Precision* (%)
Epoch 0	80.12	99.85
Epoch 1	68.24	99.73
Epoch 3	87.91	99.81
Epoch 5	94.59	99.72

TABLE 7. Performance of the GAN-optimized model.

Model	Data Source	*TPR*(%)	*Precision*(%)
Original	original	96.16	99.91
	adversarial	**86.35**	99.88
GAN-optimized	original	95.99	99.74
	adversarial	**94.59**	99.72

We also compared the performance of the GAN-optimized model on the original dataset and the adversarial set generated by MCTS-T. Table 7 shows that the GAN network optimizes the original detection mode to make the model more effective for detecting XSS adversarial examples, with TPR increased by 8.24%. While ensuring that the false alarm rate does not increase, GAN greatly improves the ability of the model to detect adversarial examples.

VI. CONCLUSION

Deep-learning-based XSS attack detection models can detect attacks effectively, but they cannot to detect adversarial examples. This paper proposed an MCTS-T adversarial example generation algorithm for XSS attacks. The proposed algorithm can generate adversarial examples for any black-box model because the MCTS-T can obtain the reward value by analyzing the statistical result of the model output. We optimize the XSS detection model with GAN to enhance its ability to defend against adversarial examples. By alternatively training the discrimination model and generating new adversarial examples, the convergent discriminant model can detect the adversarial examples to the greatest extent. The experiments show that the MCTS-T algorithm can generate effective adversarial examples to bypass deep-learning-based detectors. Moreover, the GAN-optimized XSS detection model can defend against XSS attacks and its adversarial examples. The disadvantage of MCTS-T algorithm is that it can only generate adversarial examples of XSS traffic at present. In the future, we will pay efforts to study a general adversarial example generation algorithm.

REFERENCES

[1] G. E. Hinton, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2012, pp. 1097–1105.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Boston, MA, USA, Jun. 2015, pp. 1–9.

[4] S. Pei, T. Shen, C. Gu, Z. Ning, X. Ye, and N. Xiong, "3DACN: 3D augmented convolutional network for time series data," *Inf. Sci.*, vol. 513, no. 2020, pp. 17–29, Nov. 2019.

[5] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Berlin, Germany: Springer, 2013, pp. 387–402.

[6] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2013, *arXiv:1312.6199*. [Online]. Available: https://arxiv.org/abs/1312.6199

[7] 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.

[8] A. M. Vartouni, S. S. Kashi, and M. Teshnehlab, "An anomaly detection method to detect Web attacks using stacked auto-encoder," in *Proc. 6th Iranian Joint Congr. Fuzzy Intell. Syst. (CFIS)*, Feb. 2018, pp. 131–134.

[9] K. Grosse, N. Papernot, and P. Manoharan, "Adversarial examples for malware detection," in *Proc. Eur. Symp. Res. Comput. Secur.* Cham, Switzerland: Springer, 2017, pp. 62–79.

[10] T. Chuan, Z. Yi, and Y. Yuexiang, "DroidGAN: Android adversarial sample generation framework based on DCGAN," *J. Commun.*, vol. 39, no. S1, pp. 70–75, 2018.

[11] A. Al-Dujaili, A. Huang, E. Hemberg, and U.-M. Oreilly, "Adversarial deep learning for robust detection of binary encoded malware," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, vol. 1, May 2018, pp. 76–82.

[12] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving end-to-end deep learning malware detectors using adversarial examples," 2018, *arXiv:1802.04528*. [Online]. Available: https://arxiv.org/abs/1802.04528

[13] J. Cao, Y. Hu, H. Zhang, R. He, and Z. Sun, "Learning a high fidelity pose invariant model for high-resolution face frontalization," 2018, *arXiv:1806.08472*. [Online]. Available: https://arxiv.org/abs/1806.08472

[14] X. Wang, A. Shrivastava, and A. Gupta, "A-fast-RCNN: Hard positive generation via adversary for object detection," 2017, *arXiv:1704.03414*. [Online]. Available: https://arxiv.org/abs/1704.03414

[15] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu, "HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2018.

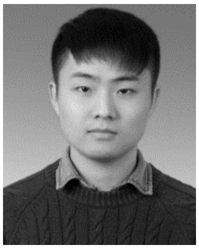
[16] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116.

[17] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014, *arXiv:1412.6572*. [Online]. Available: https://arxiv.org/abs/1412.6572

[18] S. Althubiti, W. Nick, J. Mason, X. Yuan, and A. Esterline, "Applying long short-term memory recurrent neural network for intrusion detection," in *Proc. SoutheastCon*, St. Petersburg, FL, USA, Apr. 2018, pp. 1–5.

[19] S. Dhaliwal, A.-A. Nahid, and R. Abbas, "Effective intrusion detection system using XGBoost," *Information*, vol. 9, no. 7, p. 149, Jun. 2018.

XUEQIN ZHANG received the Ph.D. degree in detection technology and automation devices from the East China University of Science and Technology (ECUST), Shanghai, China, in 2007. Since 1998, she has been with the Electrical and Communication Engineering Department, ECUST, where she is currently an Associate Professor. In 2006, she was a Visiting Scholar with the University of Wisconsin–Madison. Her research interests include information security, pattern classification, and data mining.



YUE ZHOU received the B.S. degree in microelectronics from China Jiliang University, in 2017. He is currently pursuing the M.S. degree in communication and information engineering with the East China University of Science and Technology. He has been involved in several intrusion detection projects. His research interests include intrusion detection and machine learning.

SONGWEN PEI (Senior Member, IEEE) received the B.S. degree from the National University of Defence Technology, Changsha, China, in 2003, and the Ph.D. degree from Fudan University, Shanghai, China, in 2009. He has been a Guest Researcher with the Institute of Computing Technology, Chinese Academy of Sciences, since 2011. He was a Research Scientist with the University of California at Irvine, from 2013 to 2015, and the Queensland University of Technology, in 2017. He is currently an Associate Professor with the University of Shanghai for Science and Technology. His research interests include heterogeneous multicore systems, cloud computing, and big data. He is a Senior Member of the CCF, China. He is also a Board Member of CCF-TCCET and CCF-TCARCH. He was a recipient of the Pujiang Talent of Shanghai, the Leading Talent of Suzhou, and the Shanghai Science and Technology Progress Award.

JINGJING ZHUGE received the B.S. degree in information engineering from the East China University of Science and Technology, in 2016, where he is currently pursuing the M.S. degree in signal and information processing. He has been involved in numerous network security projects. His current research interests include intrusion detection, machine learning, and neural networks.

JIAHAO CHEN received the B.S. degree from the East China University of Science and Technology, in 2016, where he is currently pursuing the M.S. degree. He has been involved in numerous network security projects. His current research interests include intrusion detection, information security, and pattern recognition.