Computers & Security 113 (2022) 102554

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

TC 11 Briefing Papers

Black-box adversarial attacks on XSS attack detection model

Qiuhua Wang[a], Hui Yang[a], Guohua Wu[a], Kim-Kwang Raymond Choo[b], Zheng Zhang[a], Gongxun Miao[c], Yizhi Ren[a,*]

[a] School of Cyberspace, Hangzhou Dianzi University, Hangzhou 310018, China
[b] Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249, USA
[c] Zhongfu Information Co., Ltd., Jinan 250101, China

ARTICLE INFO

Article history:
Received 13 February 2021
Revised 13 October 2021
Accepted 17 November 2021
Available online 24 November 2021

Keywords:
Cross-Site scripting
Adversarial attack examples
Black&white attack
Soft Q-learning

ABSTRACT

Cross-site scripting (XSS) has been extensively studied, although mitigating such attacks in web applications remains challenging. While there is an increasing number of XSS attack detection approaches designed based on machine learning and deep learning algorithms, it is important to study and evaluate the reliability and security of these approaches. In our study, focusing on machine / deep learning-based XSS attack detection approaches, we propose a fuzzing-based approach to realize "Black&White attack", in order to effectively improve the confidence coefficient of malicious samples. We also present an adversarial attack model based on Soft Q-learning, designed to generate adversarial attack examples for different XSS attack detection models using multiple strategies. Experimental results reveal that the proposed adversarial attack model generates adversarial attack examples against various XSS attack detection models, with an escape rate of over 85%. In other words, our research has implications on existing XSS attack detection models, for example in terms of effectiveness.

1. Introduction

Web applications are often the target of cyberattacks, as they are the portal to sensitive information and databases held by organizations for example by exploiting vulnerabilities in these applications using SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and other techniques. While XSS attacks have been widely studied, they remain one of the top ten web application security risks listed by the Open Web Applications Security Project (OWASP). Existing research on XSS attack detection include the use of deep learning and machine learning). However, the security and reliability of XSS attack current detection models have rarely been studied.

In the attack detection model, defending against attacks caused by adversarial attack examples and minimizing adversarial examples' impact are vital processes (Zhang et al., 2020). Adversarial attack examples (by modifying the original examples) can lead to machine learning and deep learning models' classification errors (Szegedy et al., 2013; Yuan et al., 2019). However, in the construction process, XSS scripts can only be changed based on fixed rules; otherwise, their original semantics will be destroyed. Therefore, finding and exploiting these rules to construct XSS adversarial examples is one of the current research challenges.

We observe that there are only a small number of existing studies focusing on generating XSS adversarial attack examples (Fang et al., 2019; Zhang et al., 2020), and the generated adversarial attack examples in these studies tend to have a very low escape rate (ER). Seeking to contribute to the knowledge gap, we explore the use of Soft Q-learning to generate adversarial attack examples against different kinds of XSS attack detection models, with an increased ER. A summary of our approach in this paper is as follows:

(1) We propose a new fuzzing-based approach to locate specific bypass strings. We posit that adding these bypass strings at specific positions on the XSS payload can increase the confidence coefficient (indicating the probability that the model identifies the data as malicious) of malicious examples. We also construct a FUZZ dataset containing 243,621 strings.
(2) Based on the characteristics of Hypertext Markup Language (HTML) and JavaScript syntax, we propose HTML and JavaScript bypass strategies in order to maintain the attack characteristics of attack vectors when constructing the adversarial attack examples. This will allow us to increase the confidence coefficient of XSS adversarial attack examples to a certain extent.
(3) We propose an adversarial attack model based on Soft Q-learning, which generates adversarial attack examples using

* Corresponding author.
E-mail addresses: raymond.choo@fulbrightmail.org (K.-K.R. Choo), renyz@hdu.edu.cn (Y. Ren).

https://doi.org/10.1016/j.cose.2021.102554
0167-4048/

multifold strategies and improves the escape rate of adversarial attack examples.

(4) We then demonstrate how our proposed approach can be used to generate adversarial attack examples against different XSS attack detection models such as semantic analysis-based(Chaitin, 2019), event filtering-based(Safedog, 2020), machine / deep learning-based(Fang et al., 2018) in order to demonstrate its utility.

The rest of this paper is organized as follows. Related works are introduced in Section 2. In Section 3, we provide the detailed description of our proposed adversarial attack model, and introduce how to use fuzzing-based algorithm to implement "Black&White attack", and how to construct the adversarial attack model. In Section 4, the experiment results and performance analysis are presented. Finally, we conclude the paper in Section 5.

2. Background and related work

2.1. XSS Attacks

In XSS attacks, the attacker injects malicious code into a victim's browser. When a victim browses a webpage, the malicious code will be triggered and the user's sensitive data such as cookies will be sent to the attacker. XSS attacks are usually divided into 3 types:

(1) Stored (Persistent) XSS Attack: the malicious code is stored in the server's database. When a victim accesses a vulnerable webpage, the server will retrieve the malicious code from database and send it to user, causing the user's client to execute the malicious code.

(2) Reflected (Non-Persistent) XSS Attack: in this type of XSS Attack, the malicious code is not stored in the servers database, and the attacker needs to induce the victim to click on a link containing malicious code. The victim's request to the server contains malicious code, and the HTML that the server responds contains the malicious code requested by the user.

(3) DOM-based XSS Attack: this type of attack is different from the above two types. The execution of malicious code does not require server's participation. It occurs in Document Object Model (DOM) of an HTML page. The attacker can let the malicious code execute in the HTML DOM environment to attack users.

2.2. XSS Attack detection model

Many methods based on deep learning and machine learning have been proposed for XSS attack detection in recent years. In the following paragraphs, we will review approaches relevant to this paper.

2.2.1. Machine / deep learning-based XSS attack detection approaches

Rathore et al. (2017) proposed a method based on machine learning to detect XSS attacks against social networking services (SNS). They extracted Uniform Resource Locator (URL) fields, URL lengths, links in the iframe, and malicious JavaScript codes in SNS. Then they utilized Random Forest algorithms to identify XSS attacks in the SNS environment. The results manifested that their method's accuracy rate and false alarm rate of 97.2% and 8.7%. However, they only used traditional machine learning methods, and did not consider that some methods of deep learning might learn more features and got better experimental results.

Tekerek, 2021 proposed an anomaly-based web attack detection architecture in a Web application using deep learning methods. The overall architecture is divided into data preprocessing and convolution neural network (CNN). Their research can detect various web application attacks such as SQL injection, buffer overflow, CRLF injection, XSS, and parameter tampering. The results showed that the detection accuracy reached 97.07% and the F1 score was 97.51% on the CSIC2010v2 datasets. However, attackers could use some unusual http request packets such as http chunked transfer encoding to bypass the preprocessing stage of the model.

Zhang et al. (2017) developed a URL attack detection system based on PU learning methods. They claimed their model could naturally make better use of the detected malicious URLs including XXE (XML External Entity Injection), XSS and SQL injection, etc. Their method only requires some pre-processed malicious URLs, so it is better than other supervised learning methods in a real environment. They experimented on daily-arrived URL requests in Ant Financial and the final accuracy rate reached 94.2%. However, the accuracy rate was still low. In the actual environment, the model would have a lot of false positives.

Zhou and Wang (2019) put forward an integrated learning method of XSS attack detection with domain knowledge and threat intelligence. Through this method, a set of Bayesian networks with domain knowledge and threat intelligence was used. The output nodes were sorted according to the influence of nodes in Bayesian networks making the results interpretable to end-users. When attacks increase, this method can help the security team to deal with them promptly. However, the model needed to rely on threat intelligence, and the model might face the problem that some malicious IP or domain names were not included in the threat intelligence, or the threat intelligence was no longer maintained.

Fang et al. (2018) also proposed an XSS attack detection model based on deep learning. First, XSS payloads were decoded by common decoding methods; then XSS payloads features were extracted by word2vec and trained by Long Short-Term Memory Recurrent Neural Networks (LSTM). The results showed that the detection accuracy reached 99.5%, and the recall rate was 97.9%. Therefore, the model can effectively identify XSS attacks. Akaishi and Uda (2019)used word2vec to convert the words in XSS payloads into word vectors. In the meantime, they improved the preprocessing method of vectorization and applied various machine learning algorithms to classify them. The authors claimed that CNN and SVM were the best classifiers. However, the dataset they used did not include the latest HTML5 tags and some malicious samples that were bypassed by encoding, which caused their model to not work properly when faced with adversarial samples.

Mokbal et al. (2019) proposed a dynamic feature extraction method and an XSS attack detection model based on multilayer perceptron (MLP). The MLP model was established using a real environment dataset and dynamically extracted features of XSS payloads. The detection accuracy and recall rate of the model were 99.32% and 98.35%, respectively. However, they performed JavaScript parsing and HTML parsing by traversal, which was inefficient in attack detection in the actual environment.

In machine / deep learning-based XSS attack detection approaches, the current research did not consider the influence of adversarial examples, but only trained on the known dataset. In the actual XSS attack process, the attacker can use a variety of bypass methods. If the XSS payload constructed by the attacker using some special bypass methods was not included in the training dataset of the detection model, it may make the XSS attack detection model invalid.

2.2.2. Semantic analysis-based XSS attack detection approaches

Fang et al. (2020) proposed a static detection model based on semantic analysis. Their model generated JavaScript source code into an abstract syntax tree, so that some obfuscated malicious samples could be analyzed, and the result will be convert to syntactic unit sequences. Then they used the FastText algorithm and Bi-LSTM to detect malicious JavaScript. Experiments showed that the detection precision and recall reached 0.977 and 0.974.

In XSSChop(Chaitin, 2019), its detection method is different from conventional methods based on machine learning and deep learning. XSSChop scored the results of HTML semantic analysis and JS syntax analysis of XSS payloads, which can get a high accuracy rate and a very low false positive rate. However, the method led to a low recall rate when dealing with some XSS attacks with context. Also, attackers can bypass the detection of XSSChop using parsing engines difference.

2.3. XSS Adversarial attack models

By modifying original examples, the adversarial attack examples will lead to classification errors of machine learning and deep learning models (Szegedy et al., 2013). For instance, in the field of image recognition, after adding subtle disturbances that is imperceptible to human eyes to a panda picture, the deep network will identify the panda as a gibbon with a confidence coefficient of as high as 99.3%. Similarly, an attacker can construct XSS adversarial examples to induce XSS attack detection models and give a wrong output with high confidence. The current researches on the XSS adversarial attack model are as follows:

Fang et al. (2019) proposed a DDQN-based adversarial attack model, which performed adversarial attacks on different XSS attack detection models and obtained adversarial attack examples. However, the bypass method is relatively simple, with no consideration for some bypass strategies specific to machine / deep learning-based XSS attack detection approaches. DDQN used in this paper belongs to the policy-based reinforcement learning algorithm. Therefore, only a few bypass strategies for an XSS payload produced a low ER of the proposed model.

Based on Monte Carlo Tree Search (MCTS) algorithm, Zhang et al. (2020) utilized the MCTS-T algorithm for XSS attacks and optimized the XSS attack detection model based on Generative Confrontation Network (GAN). However, only some common encoding and case-replacement were applied in the bypass method. Besides, some encoding would lead to the invalidation of attack characteristics of XSS payloads. Furthermore, the time complexity of the algorithm became very high after more escape actions.

The current researches generate low ER, which is mainly due to the following reasons:

(1) Some bypass strategies specific to machine / deep learning-based XSS attack detection approaches are not considered.
(2) Policy-based reinforcement learning is used.
(3) Simple bypass actions are used.
(4) There is only one escape stage.

In response to the above shortcomings, we have made the following improvements:

(1) Focusing on machine / deep learning-based XSS attack detection approaches, we propose a fuzzing-based approach to realize "Black&White attack" to effectively improve the confidence coefficient of malicious samples.
(2) We use maximum entropy-based reinforcement learning algorithm to discover all of the ways to perform the task.
(3) According to the HTML syntax and JavaScript syntax parsing rules, we respectively proposed more bypass actions.
(4) We divide the Black-box adversarial attack model into HTML escape stage and JavaScript escape stage which is an efficient way to bypass XSS attack detection.

2.4. Soft Q-learning algorithm

Soft Q-learning algorithm is applied as a reinforcement learning algorithm based on maximum entropy reinforcement learning (Haarnoja et al., 2017). Compared with the standard reinforcement learning algorithm, it is incentivized to explore more widely (Haarnoja et al., 2018). And maximum entropy reinforcement learning try to discover all of the ways to perform the task. In the standard reinforcement learning algorithm, the learning goal is defined by Eq. (1).

$$\pi^*_{\text{std}} = \text{argmax}_\pi \sum \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}[r(\mathbf{s}_t,\mathbf{a}_t)] \tag{1}$$

Based on the maximum entropy, the learning objective of the reinforcement learning algorithm is defined by Eq. (2)

$$\pi^*_{\text{MaxEnt}} = \text{argmax}_\pi \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_n}[r(s_t,a_t)+, \alpha\mathcal{H}(\pi(\cdot\,|s_t))] \tag{2}$$

where $\mathcal{H}(.)$ is an entropy function that brings randomness of strategy, α is the temperature parameter used to control the importance of entropy.

After adding the influence of entropy to Q-function, Soft Q-function is defined by Eqs. (3) and (4).

$$\begin{aligned} &Q^*_{\text{soft}}(\mathbf{s}_t,\mathbf{a}_t) \\ &= r_t + \mathbb{E}_{(\mathbf{s}_{t+1},\ldots)\sim\rho_\pi}\left[\sum_{l=1}^{\infty}\gamma^l\left(r_{t+l}+\alpha\mathcal{H}\left(\pi^*_{\text{MaxEnt}}(\cdot\mid\mathbf{s}_{t+l})\right)\right)\right] \end{aligned} \tag{3}$$

and V value function is:

$$V^*_{\text{soft}}(\mathbf{s}_t) = \alpha\log\int_{\mathcal{A}}\exp\left(\frac{1}{\alpha}Q^*_{\text{soft}}\left(\mathbf{s}_t,\mathbf{a}'\right)\right)d\mathbf{a}' \tag{4}$$

With the definition of state behavior value function and state value function, the expression of the optimal strategy is expressed as Eq. (5).

$$\pi^*_{\text{MaxEnt}}(\mathbf{a}_t\mid\mathbf{s}_t) = \exp\left(\frac{1}{\alpha}\left(Q^*_{\text{soft}}(\mathbf{s}_t,\mathbf{a}_t) - V^*_{\text{soft}}(\mathbf{s}_t)\right)\right) \tag{5}$$

Soft Q-Iteration:
Eq. (3) is updated to :

$$Q^*_{\text{soft}}(\mathbf{s}_t,\mathbf{a}_t) = r_t + \gamma\mathbb{E}_{\mathbf{s}_{t+1}\sim p_{\mathbf{v}}}\left[V^*_{\text{soft}}(\mathbf{s}_{t+1})\right] \tag{6}$$

The Q function and V function are iteratively updated in the following ways:

$$Q_{\text{soft}}(\mathbf{s}_t,\mathbf{a}_t) \leftarrow r_t + \gamma\mathbb{E}_{\mathbf{s}_{t+1}\sim p_s}[V_{\text{soft}}(\mathbf{s}_{t+1})], \forall\mathbf{s}_t,\mathbf{a}_t \tag{7}$$

$$V_{\text{soft}}(S_t) \leftarrow \alpha\log\int_{\mathcal{A}}\exp\left(\frac{1}{\alpha}Q_{\text{soft}}\left(\mathbf{s}_t,\mathbf{a}'\right)\right)d\mathbf{a}', \forall\mathbf{s}_t \tag{8}$$

Theoretically, if recursion is continuously conducted, Q and V will be eventually converged to Q^*_{soft} and V^*_{soft}.

The standard reinforcement learning algorithm continually interacts with the environment to find an optimal bypass strategy. At the same time, Soft Q-learning is a reinforcement learning algorithm based on maximum entropy that can fully explore the environment and find more strategies to complete the required tasks. Therefore, when investigating the adversarial attack examples, the reinforcement learning algorithm based on maximum entropy has more advantages than the reinforcement learning algorithm based on maximum reward.

2.5. Deep learning algorithm

In the XSS attack detection model, we use MLP(multilayer perceptron) and LSTM(Long Short-Term Memory) deep learning algorithm as our classification algorithm.

2.5.1. MLP Algorithm

In the MLP algorithm, as shown in the Fig. 1 the network structure is divided into input layer, hidden layer and output layer. The learning process of MLP is divided into two parts: forward propagation and backward propagation. In the forward propagation, data information enters from the input layer, processed by each hidden

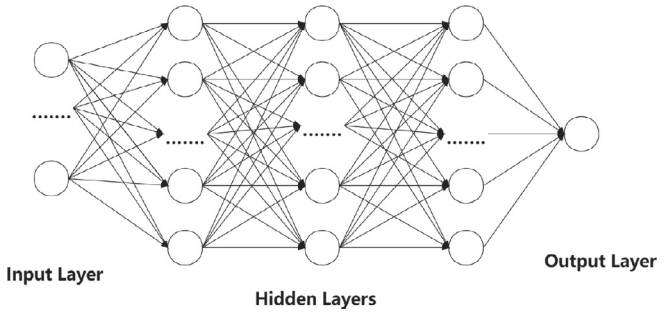

Fig. 1. MLP Model Structure.

layer neural unit, and then transmitted to the output layer. If the result obtained in the output layer does not match the expectation, the weight correction is performed through backpropagation algorithm. In the process of backward propagation, the weight parameters of the network are modified layer by layer, so that the neural network is trained through forward propagation and backward propagation.

For the input samples x_i, the network weight of the hidden layer is w_{ih} the input of the hidden layer is Eq. (9):

$$a_h = \sum_{i=1}^{I} w_{ih} x_i \tag{9}$$

The output of the first layer of the hidden layer is Eq. (10):

$$b_h = f(a_h) \tag{10}$$

Function f is a nonlinear activation function. After calculating the conduction from the input layer to the first hidden layer, the calculation method of the remaining hidden layer is similar, and h_l is used to represent the number of units in layer l. The output of the every layer of the hidden layer is Eq. (12):

$$a_h = \sum_{h'=1}^{hl-1} w_{h'h} b_{h'} \tag{11}$$

$$b_h = f(a_h) \tag{12}$$

$$a = \sum_{h'} w_{h'h} b_{h'} \tag{13}$$

$$y = f(a) \tag{14}$$

For the output layer, we use logistic regression to achieve binary classification. In Eq. (14), y is the confidence coefficient the probability that the MLP model identifies the data as malicious.

2.5.2. LSTM Algorithm

Fig. 2 shows how LSTM works. On the basis of RNN(Zaremba et al., 2014), LSTM adds three gates: the forget gate f_t, the input gate i_t, the cell gate g_t and the output gate o_t. It overcomes the shortcomings of excessive influence of RNN weights, vanishing gradient problem and exploding gradient problem. The LSTM network can converge better and faster, and can effectively improve the prediction accuracy.

In Eq. (15) W_f is the weight matrix of the forget gate; b_f is the bias of the forget gate. σ is the sigmoid function.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{15}$$

W_i is the input gate weight matrix, b_i is the bias term of the input gate.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{16}$$

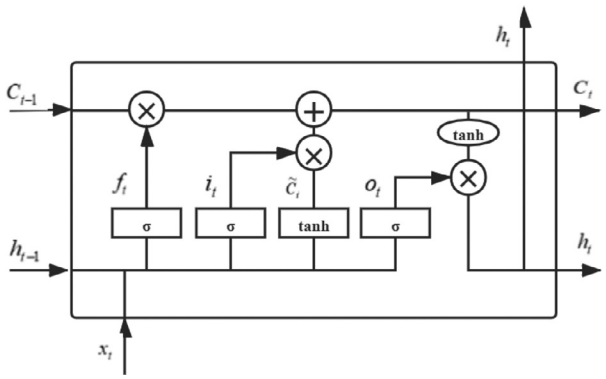

Fig. 2. LSTM Model Structure.

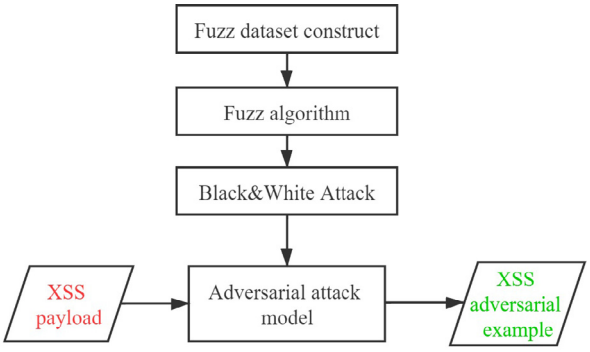

Fig. 3. The Framework Of Our Proposed Method .

$$g_t = \tanh(W_g \cdot [h_{t-1}, x_t] + b_g) \tag{17}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{18}$$

$$h_t = o, \odot \tanh(c_t) \tag{19}$$

Eq. (18) calculates the output of the output gate. Eq. (19) finally obtains LSTM output.

3. Proposed method

3.1. Overview

Fig. 3 presents the framework of our proposed method. First, in order to obtain the benign samples, we use crawler technology and existing dataset to construct a fuzz dataset. Next, based on the fuzz dataset, we use fuzz algorithm to implement Black&White attack. The Black&White attack can effectively improve the confidence coefficient of malicious samples and it will be used as a key bypass strategy for the adversarial attack model. Then, an XSS payload which can be recognized by the XSS attack detection model is used as the input of the adversarial attack model. Finally, after the adversarial attack model processes the XSS payload by bypass strategies, it will output an XSS adversarial example which retains the attack characteristics and can bypass the XSS attack detection model.

3.2. FUZZ Algorithm and black&white attack

Our current research reveals that for XSS attack detection model which is based on deep learning and machine learning, adding some specific strings to some particular positions in XSS

Alexa top1000 parameters:	jQuery112400655284765504684 5_1602697106492 jQuery112406554260661268738_1602724289306 wp_portlet_css__0.0%3Ahead_css osaka-sakaishi-miharaku MediaWiki:Stylesheets-Tabs.css AuthPortalPoolUS .....
English words top10w:	sense close subject turn town .....
Regular strings:	abcdefg abc%20abcd%20abcdc abc%20abcd%20abcdc%20abcdcfgh 123456789 987654321 .....

Fig. 4. FUZZ Dataset Examples.

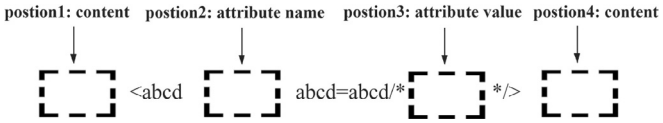

Fig. 5. Benign XSS FUZZ Example.

payload can effectively increase the confidence coefficient. It also makes the classification algorithm produce wrong classification results. These strings are usually the strings with some regularity, some normal English words, and some benign examples in the XSS attack detection model's training dataset. We call this type of adversarial attack as "Black&White attack". Due to unknown benign samples trained by XSS attack detection models under the Black-box environment, the FUZZ dataset is constructed from the following three aspects to obtain benign examples:

(1) We use web crawler technology to crawl the Alexa (Cooper, 2020) top 1000 website in depth first with a maximum depth of 4 layers. The parameters in the crawled URL are segmented, obtaining 142,621 request parameters.
(2) 100,000 commonly used English words.
(3) 1000 strings are generated according to specific rules such as alphabetical order rules and numerical order rules.

The generated FUZZ dataset is considered as an input to the FUZZ algorithm. Some examples are shown in Fig. 4. The XSS attack detection model usually treats these data as normal data. But there are many such normal data, we need to know where to add such normal data in the XSS payload to have a greater impact on the XSS attack detection model, and to ensure that such changes will not affect its attack characteristics.

So, in order to get the influence of fuzz dataset on XSS attack detection model, string of benign example with some XSS characteristics is constructed (Fig. 5). Here, the XSS characteristics of an XSS sample includes tag name, attribute name, attribute value, event and malicious JavaScript. The benign example we constructed contains tag name event and JavaScript, but these characteristics are all random strings, so the benign example only has some structural characteristics of malicious XSS samples. The confidence coefficient prediction results of LSTM-based and MLP-based XSS attack detection model for this example are 0.6911, 0.4892. The confidence coefficient represents the probability that the XSS attack detection model identifies the data as malicious, and it is calculated by the classification algorithm used by the LSTM-based and MLP-based XSS attack detection model. The LSTM-based and MLP-based XSS attack detection model will be introduce in Section 4.2.

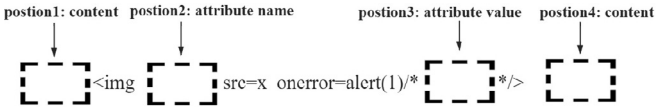

Fig. 6. Malicious XSS FUZZ Example.

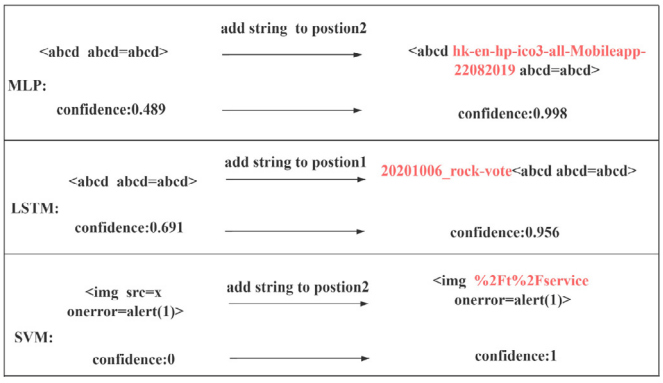

Fig. 7. Black& White Attack Examples.

In binary classifiers like SVM, the payload of FUZZ is changed into malicious examples. Fig. 6 displays a string of malicious XSS fuzz example constructed in this paper. In binary classifiers, the special string exerting the most significant influence on the model can be found.

Next, the FUZZ algorithm is applied to the four positions, 1 to 4, of FUZZ examples as depicted in Fig. 5 and Fig. 6. Adding strings in these four positions preserve the HTML semantics and JavaScript semantics of the example. Therefore, the attack features of XSS payload are not destroyed. Finally, all of the obtained results are sorted. The algorithm is illustrated in Algorithm 1 .

Algorithm 1: Fuzz Algorithm.

input : WORDS[]:list fuzz data;DM[]: list XSS detection model;P[]: list fuzz postions;payload: a fuzz example;
output: RES[]:fuzz results

```
WORDS[] ← list fuzz words;
DM[] ← list XSS detection model;
RES[] ← results;
POS[] ← list fuzz postions;
payload ← a fuzz example;
foreach fuzz postion p in POS do
    foreach each XSS detection model dm in DM do
        foreach each fuzzword in WORDS do
            Compute the confidence coefficient of the model
            dm when the fuzzword is added to the position p
            of the payloadUpdate RES
        end
    end
end
```

After extensive FUZZ tests, the top 10 strings with the most significant interference on each detection model are detected. Adding these strings to XSS payloads can realize Black&White attack on the XSS attack detection model. Some of the attack results are displayed in Fig. 7. Generally, the XSS attack detection model will first preprocess the payload and then classify it. When the model preprocesses these adversarial examples, it will get many benign features, so the Black&White Attack will increase the confidence of the examples. And the Black&White attack will be an important

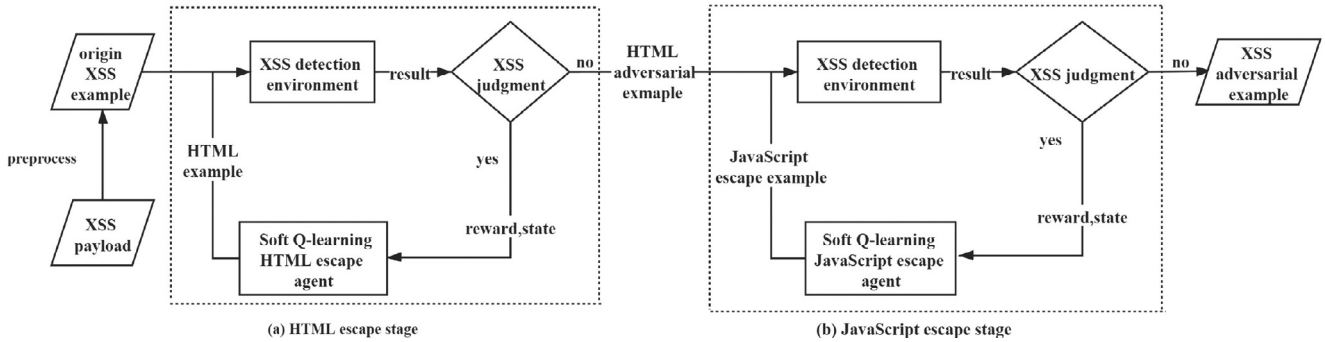

Fig. 8. Adversarial Attack Model.

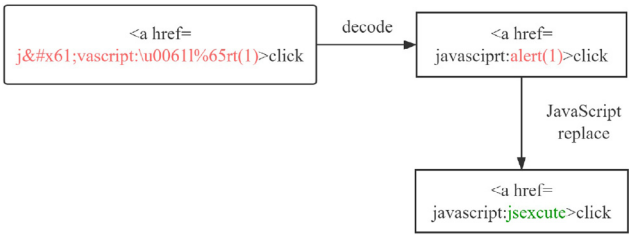

Fig. 9. Preprocess.

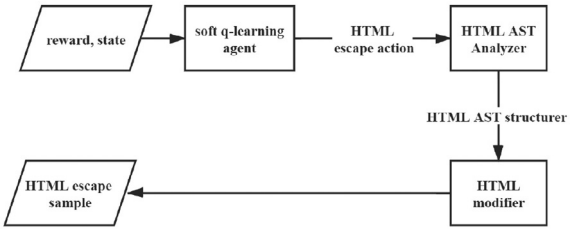

Fig. 10. HTML Escape Agent.

bypass strategy for the adversarial attack model introduced in the next section.

3.3. Black-box adversarial attack model

Fig. 8 displays the framework of the adversarial attack model designed in this study. In a real-world environment, an efficient way for an attacker to bypass XSS attack detection is to divide the adversarial attack into HTML escape stage and JavaScript escape stage, hence, we mainly divide the Black-box adversarial attack model into HTML escape stage and JavaScript escape stage. Our aim is to make the adversarial attack model clearly know whether the HTML part or the JavaScript part of the sample is detected during the attack, and then choose the corresponding bypass strategies. Before the HTML escape stage and the JavaScript escape stage, we also design a preprocess stage. In the preprocess stage, most HTML examples that can trigger JavaScript are collected as XSS payloads and preprocessed as origin XSS examples. At the HTML escape stage, the original XSS example is applied to the XSS detection environment, if the XSS detection environment can identify the examples as malicious, the reward and state will be the output to Soft Q-learning HTML escape agent. Soft Q-learning HTML escape agent selects escape actions according to the state and learns according to the reward feedback output by the XSS detection environment. At the end of the HTML escape stage, the HTML escape agent will output an HTML adversarial attack example that can execute JavaScript. However, the example does not contain any malicious JavaScript. At the JavaScript escape stage, malicious JavaScript example which can escape detection can be found based on HTML adversarial attack examples. After the above steps, XSS adversarial attack examples can be obtained.

3.3.1. Preprocess

At the preprocess stage, URL, HTML, and JavaScript decoding are performed on XSS payloads; then, the original JavaScript malicious example is preprocessed, and the JavaScript malicious example in XSS payloads is replaced with a normal string. The results after processing are taken as origin XSS examples. Fig. 9 shows the process of preprocessing an XSS payload. After the preprocessing, XSS payload becomes an example that can execute JavaScript but does not contain any malicious code.

3.3.2. HTML escape stage

After passing the origin XSS example to the HTML escape stage, the Soft Q-learning HTML escape agent can bypass this example and output HTML adversarial attack example without malicious JavaScript.

(1) Black-box XSS detection environment
Compared with the White-box environment, we can only know the confidence coefficient of sample in the Black-box environment. Therefore, the XSS detection environment will only output state and reward for the input. State represents the structure information of the current payload and the reward is defined in Eq. (20) as follows:

$$Reward = (result_{new} - result_{old}) * score \tag{20}$$

where $result_{new}$ and $result_{old}$ represent the confidence coefficient of examples after and before bypassing the escape agent in the XSS detection environment, respectively. The score is a fixed reward value. Then we construct a total of five XSS detection environments as follows:

1. Features are extracted using word2vec based on the data preprocessing method of Deepxss (Fang et al., 2018), an LSTM based on deep learning is developed.
2. XSS attack detection model based on MLP.
3. XSS attack detection model based on SVM.
4. XSSChop based on semantic analysis.
5. SafeDog based on event filtering.

(2) Soft Q-learning-based HTML escape agent
HTML escape agent escapes the HTML part of XSS payloads. The model of the HTML escape agent is shown in Fig. 10. When the environment feeds back a set of reward and state, the Soft Q-learning agent will save them and use Eqs. (6) and (7) to update Q_{soft} and V_{soft}. After multiple rounds of training Q_{soft} will converge to Q^*_{soft} and V_{soft} converge to V^*_{soft}. The Soft Q-learning agent will select the best HTML escape action(which will be described later) according to the Q^*_{soft} and current state. To modify HTML examples conveniently and accurately, an HTML AST

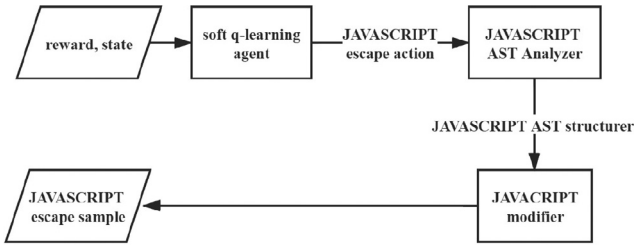

Fig. 11. JavaScript Escape Agent.

Analyzer is utilized. It parses the HTML examples into HTML AST structure, with Parse5(Anko, 2020) HTML parsing engine. Later, the HTML modifier modifies the HTML examples by the HTML AST Structure according to the escape action to generate the HTML adversarial example.

(3) HTML escape action

According to the HTML syntax and parsing process, the following bypass actions for XSS payloads are proposed. The bypass actions are mainly divided into string substitution, character encoding, and string addition.

1. String substitution:
   (a) Separator substitution: the separators between the tag name and attribute name are substituted with %20, %09, %0a, %0d, and %0c;
   (b) Case substitution: the string of attribute name or tag name is subjected to case substitution.
2. Character coding:
   (a) HTML encoding;
   (b) Add zero to the middle of HTML entity codes;
   (c) Some special characters or HTML codes are substituted with HTML entity coding names;
   (d) Convert the decimal and hexadecimal of HTML entity code to each other.
3. String addition:
   (a) Add %0d, : 	
 %09 and %0a between "javascript:";
   (b) Add %10-%1f before "javascript:";
   (c) Add random strings before or after the tag;
   (d) Add the corresponding special string that can effectively increase the malicious example confidence coefficient (Black&White attack). As stated previously in Section 3, the FUZZ algorithm can realize the Black&White attack;
   (e) Add "%26%2362%3B" string between attributes, which affects the decoder of defense model using WAF decoding mechanism;
   (f) Add %0a, %0c, %0d, %09, before attributes.

3.3.3. Javascript escape stage

The XSS detection environment applies the same XSS attack detection model at both JavaScript escape and the HTML escape stages. The difference is that the HTML part of the payload is bypassed at the HTML escape stage, while the part of malicious JavaScript is bypassed at the JavaScript escape stage. After the two steps of bypassing, XSS adversarial attack examples will be generated.

As shown in Fig. 11, the JavaScript escape agent performs JavaScript escape on the HTML adversarial example. The training process of Soft Q-learning agent is similar to that of HTML escape stage. The Soft Q-learning agent makes an optimal escape action according to Q^*_{soft} and current JavaScript state. To modify the JavaScript more conveniently and accurately, the JavaScript AST analyzer is used to parse JavaScript examples into JavaScript AST Structure, and @babel/parser (Babel, 2020) is applied as the JavaScript parsing engine. Later, the JavaScript modifier uses JavaScript AST structure in the form of parsed AST syntax tree structure as an input. According to the Soft Q-learning agent's action, the modifier modifies JavaScript examples to generate XSS adversarial attack example.

JavaScript escape action:

According to JavaScript syntax parsing rules and encoding and decoding rules, bypass strategies are proposed for JavaScript in XSS payloads as follows:

1. Add %0a, %0b, %0c, %0d, %09 before (, [;
2. JavaScript is randomly encoded into HTML entities;
3. Unicode encoding of the JavaScript code;
4. Add a random string at the comment;
5. URL encoding of the string "javascript";
6. Add the corresponding special string at the comment that can effectively increase the malicious example confidence coefficient (Black&White attack). As stated previously in Section 3, the FUZZ algorithm can realize the Black&White attack;
7. Jsfuck encoding;
8. JavaScript emoticons encryption.

Through the escape at the HTML and JavaScript escape stages, malicious XSS payloads' attack characteristics are not changed. And some characteristics of malicious examples may become utterly different from those before the escape. When some characteristics are changed, the classification of these examples will become inaccurate.

4. Experiments and discussions

4.1. The dataset

The dataset in this experiment is divided into three parts:

(1) Training dataset of the XSS attack detection model. We use the dataset of DeepXSS (Fang et al., 2018), which contains 40,637 malicious XSS examples and 31,407 normal requests on XSSED, both are utilized in the training.

(2) XSS payloads of adversarial attack models. In the real network environment, the attacker and the defender's information is not equal. The defender can not fully know what kinds of tags and bypass techniques the attacker will use. Attackers also can not fully recognize which examples the defenders have intercepted, which interception strategies they have adopted, or which bypass techniques they have intercepted. Attackers should try as many HTML tags as possible. Hence, XSS payloads containing most HTML tags and corresponding trigger JavaScript events are collected from the PortSwigger XSS cheat sheet (portswigger, 2020).

(3) Original malicious JavaScript examples of attack model. We find that for some detection models based on semantic analysis, the inconsistency of JavaScript parsing engines may cause the XSS attack detection model to parse JavaScript incorrectly, which can be applied for bypassing detection. We construct some malicious examples according to ECMAScript 2020 syntax (International, 2020), and extracted 20 malicious JavaScript examples from XSSED (KF, 2012).

4.2. Evaluation criteria and experiment results

In order to evaluate the effectiveness of the adversarial attack model, we use DR and ER to evaluate.

(1) DR(detection rate): DR is the ratio of the number of malicious XSS examples that are still judged as malicious by the XSS attack detection model to the total examples.

(2) ER(escape rate): ER is the ratio of the number of malicious XSS examples judged as benign by the XSS attack detection model

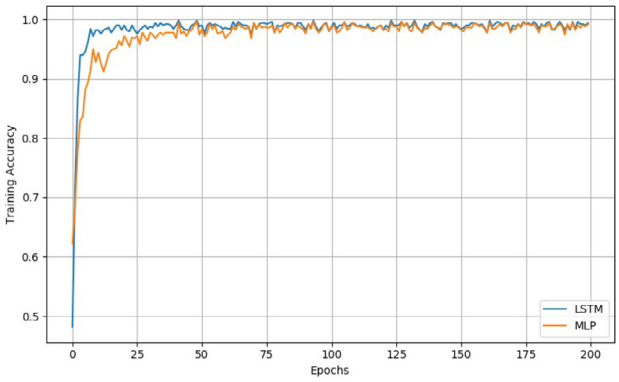

Fig. 12. Accuracy Of The Training Dataset.

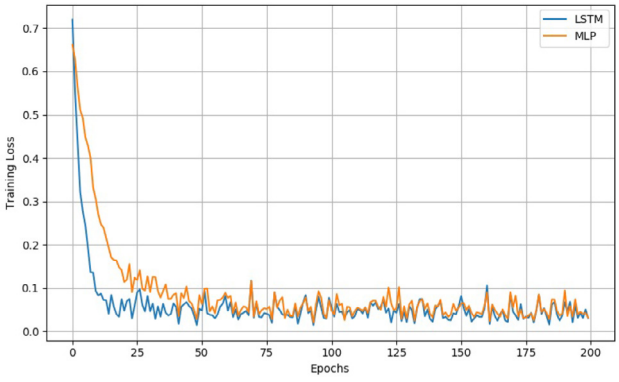

Fig. 13. Loss Of The Training Dataset.

after being bypassed by the adversarial attack model to the total examples.

To examine the adversarial attack model's effectiveness, XSS attack detection models are established based on deep learning LSTM and MLP. First, referring to the preprocessing process of DeepXSS (Fang et al., 2018), we use word2vec to generate word vectors from XSS samples. Then the word vector is used as the input of MLP and LSTM model. The model will be trained based on the word vector and the corresponding label, in every iteration step, the results of training accuracy and training loss in different deep learning models are calculated in Fig. 12 and Fig. 13 based on the test dataset. As an intuitive suggestion, after the first 50 rounds of training, the training accuracy and the training loss will rapidly converge. And after that, the training accuracy and the training loss in the straight corridor stabilize. It should be noted that the convergence of both the loss and the accuracy of the LSTM model are more efficient and apparent than that of MLP. After 200 rounds of training, the LSTM model's loss is 5.4%, and the ACC reaches 98.4%. While, the loss of the MLP model is 7.23%, and the ACC reaches 98%. In addition, we also build an XSS attack detection model based on SVM, and use two popular commercial XSS attack detection models: SafeDog and XSSChop. After testing these five models with the dataset used by DeepXSS, the results are shown in Table 1.

Table 1 reveals that under the same dataset, the accuracy, recall rate, and F1 score of MLP and LSTM models are higher than those of DeepXSS. SVM, LSTM, MLP, SafeDog, and XSSchop precision are collectively over 99%, proving that these XSS attack detection models can effectively detect XSS attacks without considering adversarial attack examples.

Table 1
Comparative Experiment Of Different XSS Attack Detection Model.

Classifier	Precision	Recall	F1	Accuracy
XSSChop	0.9999	0.8243	0.9036	0.8494
MLP	0.9983	0.9837	0.9910	0.9908
LSTM	0.9991	0.9842	0.9916	0.9914
SafeDog	0.9972	0.9286	0.9617	0.9366
SVM	0.9984	0.9776	0.9879	0.9877
DeepXSS	0.995	0.979	0.987	-

4.3. Adversarial attack experiment results

We input each string of 4.1.(2) dataset and RLXSS dataset into the adversarial attack model, and adversarial attack examples are generated for each XSS attack detection model. After bypassing all origin XSS payloads through the adversarial attack model, the obtained DR and ER are shown in Table 2.

The experimental results illustrate that for each XSS attack detection model, the ER of the adversarial attack examples constructed in this study is over 85%, far exceeding RLXSS(Fang et al., 2019) results. Compared with the RLXSS dataset, our dataset can improve the ER of the adversarial attack model. In other words, all of the five detection models, LSTM-based, MLP-based, SVM-based, SafeDog and XSSChop, do not perform well when facing the adversarial attack model constructed in this paper. This brings challenges to the current XSS attack detection model.

Fig. 14 shows some adversarial attack examples:

(1) Fig. 14(a), MLP's confidence coefficient in origin XSS example is 0.01%. It reaches 99.84% after bypassing by the adversarial attack model constructed in this paper, in which the red part is the malicious JavaScript in XSS example. Fig. 14(b) and Fig. 14(c) also show XSS adversarial attack examples generated for LSTM-based and SVM-based models. In these XSS attack detection models, adding the feature of benign samples to the malicious XSS payload, or changing the original features of the malicious XSS payload, may cause wrong output to this kind of XSS attack detection model.
(2) Fig. 14(d) displays the adversarial attack example against SafeDog. Some strings are added to malicious JavaScript, and using an unusual HTML tag could successfully cause the wrong output of the model.
(3) Fig. 14(e) shows the adversarial attack example for XSSChop. The adversarial attack model can bypass XSSChop's detection using the inconsistency of JavaScript parsing engines.

4.4. Discussions

4.4.1. Analysis of adversarial results

For the three types of XSS detection models, we will separately explain why this model will cause the wrong output of XSS attack detection model.

(1) For the deep learning-based XSS attack detection model, the characteristics of the benign samples are added to the malicious samples. When this type of XSS attack detection model performs feature extraction, the characteristics of the benign samples will be extracted to increase the confidence of the malicious samples. In Fig. 14(a) and 14(b), the adversarial attack model will add or modify strings in the malicious sample and retain the attack characteristics, while changing the original characteristics of the malicious sample. When the original characteristics of the malicious sample are changed, the XSS attack detection model produces wrong classification results.
(2) For the event filtering-based XSS attack detection model, adding some special characters such as "=", "<", ">" to the malicious XSS sample may destroy the XSS attack detection

Table 2
Evaluating results.

Model	Our method (Our dataset)		Our method (RLXSS dataset)		RLXSS	
	DR	ER	DR	ER	DR	ER
SafeDog	0.031	**0.969**	0.057	0.943	0.859	0.141
XSSChop	0.058	**0.942**	0.879	0.121	0.907	0.093
LSTM	0.138	**0.862**	0.241	0.759	0.9175	0.0825
MLP	0.047	**0.953**	0.119	0.881	-	-
SVM	0.141	**0.859**	0.198	0.802	-	-

model's process of extracting events and attributes of the sample, thereby making this type of model produce wrong classification results. In Fig. 14(c), the adversarial attack model adds lots of special characters to the attribute value, and JavaScript is deformed to bypass detection.

(3) For the semantic analysis-based XSS attack detection model, our model will use parsing engines difference to bypass. When an attacker uses feature of the latest ECMAScript syntax to construct an XSS payload, it may make XSS attack detection model produce wrong classification results by different JavaScript parsing engines. In Fig. 14(d), the adversarial attack model can bypass the detection with only a slight modification to JavaScript. Because in the latest version of ECMAScript's syntax, "1_2" is a correct syntax and represents "12", while in the old version of ECMAScript, "1_2" is a wrong syntax. Therefore, the adversarial example can bypass detection.

We analyzed why the dataset we constructed can improve ER. We have collected as many HTML tags as possible. The training dataset used by the XSS attack detection model may not contain the HTML tags we collected, which will increase ER to a certain extent. At the same time, we have collected some malicious JavaScript samples that support the latest ECMAScript grammar. AS we can see from the adversarial results of XSSchop in Table 2, using RLXSS dataset will make our method output a very low ER. When the new dataset is used, the ER is greatly improved.

4.4.2. Suggestions for defending against adversarial attack

When building an XSS attack detection model, we will give the following suggestions to defend against this XSS adversarial attack.

(1) For the deep learning-based XSS attack detection model, some meaningless attributes and strings should be removed in the data preprocessing stage. The model need to find out which part of the payload executes JavaScript and use the algorithm to classify the JavaScript part, otherwise the algorithm will learn a large number of invalid features and cause wrong output.
(2) For the event filtering-based XSS attack detection model, the builder should improve the filtering rules, deal with special characters, and support as many tags and events as possible.
(3) For the semantic analysis-based XSS attack detection model, the builder should maintain the JavaScript parsing engine to support the latest ECMAScript syntax. And due to the flexibility of JavaScript syntax, special encoding should be processed.

4.4.3. Severity level of the attacks

The severity level of the XSS attack detection model affected by the attacks is high. Here are the reasons:

(1) In Table 1, according to precision, recall, F1 rate, and accuracy, it is shown that these XSS attack detection models can effectively detect normal XSS attacks. However, according to the attack results of Table 1, it can be seen that the ER of each XSS attack detection model exceeds 0.85. That is to say, 99% of the malicious samples will be detected before the adversarial attack, but after these XSS attack detection models are bypassed by the adversarial attack model, at least 85% of the samples will bypass the detection. As a result, these detection model become no longer valid.
(2) As can be seen from Fig. 14(a), the confidence of the model for the XSS sample before the bypass is 99.99%. However, after bypass, the detection model has 99.84% confidence that the adversarial sample is regarded as normal data. This completely deceives the detection model.

4.4.4. Analysis of time complexity

In our work, the majority of the time consuming are caused by the training process of the soft Q-learning Model. Hence, we mainly focus on discussing the time complexity of soft Q-learning.

A Markov decision process is a four tuple (S, A, P, R), where $|S|$ is the state space, $|A|$ is the action space, P is the state transition probability, and R is the instantaneous reward. Suppose there are $|E|$ epochs and $|T|$ time steps in soft Q-learning, we can further make the following assumptions:

(1) To simplify the sampling process of an action, we can assume ξ follows a discrete normal distribution $N(0, I)$, and the number of items in $N(0, I)$ is $|K|$.
(2) The batch size in experience replay of soft Q-learning is $|M|$.

Therefore, the time complexity of finite time step soft Q-learning is comprised of three parts:

(1) The first part is collecting experience, the time complexity is $|E| \cdot |T| \cdot |A| \cdot |S|$.
(2) The second part is updating soft Q-function parameters, the time complexity is $|E| \cdot |T| \cdot |M| \cdot |A|$.
(3) The third part is updating policy, the time complexity is $|E| \cdot |T| \cdot |M| \cdot |K|$.

In the worst condition, the Time Complexity of finite time step soft Q-learning is:

$$\max(|E| \cdot |T| \cdot |S| \cdot |A|, |E| \cdot |T| \cdot |M| \cdot |A|, |E| \cdot |T| \cdot |M| \cdot |T|)$$

As a comparison in Zhang et al. (2020), the majority of time consuming is triggered by the of Monte Carlo tree search (MCTS) algorithm and training process.

The implementation of MCTS on Reinforcement learning is widely studied by many researchers (Ding et al., 2019), let $|T|$ denote the depth of the tree structure, $|S| \cdot |A|$ denote the average number of child nodes at each layer of the tree structure. $|M|$ is the number of entries in D that is used to evaluate the action value, where $|D|$ is the replay memory set of MCTS.

Time Complexity of finite time step MCTS is below:

(1) The Selection step has the time complexity $|S| \cdot |A|$.
(2) The Expansion step has the time complexity of $|S| \cdot |A| \cdot |M|$.
(3) The Simulation step has the time complexity $|S| \cdot |A| \cdot |M|$.
(4) The Backpropagation step has the time complexity of $|T| \cdot |M|$.

Therefore, the worst of the time complexity of MCTS is $(|S| \cdot |A| + |T|) \cdot |E| \cdot |M|$.

The overall time complexity of their algorithm is calculated by given following assumptions:

(1) The size of the iteration steps is $|G|$.
(2) The size of the time horizon is $|T|$.

Fig. 14. Adversarial Examples.

Therefore, the worst of the time complexity is:

$$(|S| \cdot |A| + |T|) \cdot |E| \cdot |M| \cdot |G| \cdot |T|$$

Therefore, in terms of time complexity, the Soft Q-learning model significantly outperforms the model in Zhang et al. (2020).

5. Conclusion

For diverse XSS detection models, an adversarial attack model based on Soft Q-learning is put forward in this paper. Black&White attack adversarial attack method is proposed for XSS attack detection models based on machine learning and deep learning,

which could effectively interfere with XSS attack detection models' classification ability. In the adversarial attack model framework, adversarial attacks are divided into the HTML escape stage and JavaScript escape stage. Following two stages of escape, XSS adversarial attack examples are finally generated. It is discovered from the experimental results that the ER of the XSS attack detection model based on machine learning, deep learning, event filtering, and semantic analysis reach more than 85%. It proves that the XSS attack detection model's output based on these methods is no longer reliable when facing adversarial attacks. Hence, designing approaches to mitigate such adversarial attacks is one potential future extension.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Qiuhua Wang: Supervision, Writing – review & editing. **Hui Yang:** Software, Data curation, Writing – original draft. **Guohua Wu:** Writing – review & editing, Validation. **Kim-Kwang Raymond Choo:** Conceptualization, Methodology, Writing – original draft. **Zheng Zhang:** Software, Validation. **Gongxun Miao:** Visualization, Investigation. **Yizhi Ren:** Conceptualization, Methodology, Validation.

Acknowledgements

This research was supported by Zhejiang Provincial Natural Science Foundation of China (Grant No. LY19F020039), Zhejiang Province Key Field R&D Program Project (Grant No. 2022C03174). The work of K.-K. R. Choo was supported only by the Cloud Technology Endowed Professorship.

References

Akaishi, S., Uda, R., 2019. Classification of XSS Attacks by Machine Learning with Frequency of Appearance and Co-occurrence. In: 2019 53rd Annual Conference on Information Sciences and Systems (CISS). IEEE, pp. 1–6.
Anko, 2020. Parse5: HTML parsing/serialization toolset for Node.js. https://github.com/inikulin/parse5.
Babel, 2020. babel/parser. A JavaScript parser: HTML parsing/serialization toolset for Node.js. https://www.npmjs.com/package/@babel/parser.
Chaitin, 2019. XSSChop: XSS detection engine. https://xsschop.chaitin.cn/#.
Cooper, K., 2020. Alexa, alexa-st at ic. https://www.alexa.com/.
Ding, P., Liu, G., Zhao, P., Liu, A., Li, Z., Zheng, K., 2019. Reinforcement Learning Based Monte Carlo Tree Search for Temporal Path Discovery. In: 2019 IEEE International Conference on Data Mining (ICDM). IEEE, pp. 140–149.
Fang, Y., Huang, C., Su, Y., Qiu, Y., 2020. Detecting malicious javascript code based on semantic analysis. Comput Secur 93, 101764.
Fang, Y., Huang, C., Xu, Y., Li, Y., 2019. RLXSS: Optimizing XSS detection model to defend against adversarial attacks based on reinforcement learning. Future Internet 11 (8), 177.
Fang, Y., Li, Y., Liu, L., Huang, C., 2018. DeepXSS: Cross site scripting detection based on deep learning. In: Proceedings of the 2018 International Conference on Computing and Artificial Intelligence, pp. 47–51.
Haarnoja, T., Pong, V., Zhou, A., Dalal, M., Abbeel, P., Levine, S., 2018. Composable deep reinforcement learning for robotic manipulation. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp. 6244–6251.
Haarnoja, T., Tang, H., Abbeel, P., Levine, S., 2017. Reinforcement learning with deep energy-based policies. In: International Conference on Machine Learning. PMLR, pp. 1352–1361.
International, E., 2020. ECMAScript® 2020 language specification 11th edition (June 2020). https://www.ecma-international.org/ecma-262/.
KF, D., 2012. XSSed: XSS attacks information. http://www.xssed.com/archive.
Mokbal, F.M.M., Dan, W., Imran, A., Jiuchuan, L., Akhtar, F., Xiaoxi, W., 2019. MLPXSS: An integrated XSS-Based attack detection scheme in web applications using multilayer perceptron technique. IEEE Access 7, 100567–100580.
portswigger, 2020. Cross-site scripting(XSS) cheat sheet. https://portswigger.net/web-security/cross-site-scripting/cheat-sheet.
Rathore, S., Sharma, P.K., Park, J.H., 2017. XSSClassifier: An efficient XSS attack detection approach based on machine learning classifier on SNSs. J Infor Proc Syst 13 (4).
Safedog, 2020. Safedog: web attack detection engine. http://www.safedog.cn/.
Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R., 2013. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199.
Tekerek, A., 2021. A novel architecture for web-based attack detection using convolutional neural network. Comput Secur 100, 102096.
Yuan, X., He, P., Zhu, Q., Li, X., 2019. Adversarial examples: attacks and defenses for deep learning. IEEE Trans Neural Netw Learn Syst 30 (9), 2805–2824.
Zaremba, W., Sutskever, I., Vinyals, O., 2014. Recurrent neural network regularization. Eprint Arxiv.
Zhang, X., Zhou, Y., Pei, S., Zhuge, J., Chen, J., 2020. Adversarial examples detection for XSS attacks based on generative adversarial networks. IEEE Access 8, 10989–10996.
Zhang, Y.-L., Li, L., Zhou, J., Li, X., Liu, Y., Zhang, Y., Zhou, Z.-H., 2017. Poster: A pu learning based system for potential malicious url detection. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 2599–2601.
Zhou, Y., Wang, P., 2019. An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence. Comput Secur 82, 261–269.

Qiuhua Wang received her B.S. and M.S. degrees in communication engineering from Liaoning Technical University, Fuxin, China, in 2000 and 2003, respectively. She received her Ph.D. degree in communications and information systems from Zhejiang University, Hangzhou, China, in 2013. She is currently an Associate Professor at the School of Cyberspace, Hangzhou Dianzi University. Her current research interests include network security, security issues in wireless networks, key management and physical layer security.

Hui Yang is currently a M.S student at the School of Cyberspace, Hangzhou Dianzi University. His research interests include adversarial machine learning and web security.

Guohua Wu received the B.S. degree from the Shandong University of Technology, Jinan, China, in 1992, the M.S. degree from the National Institute of Metrology, Beijing, China, in 1995, and the Ph.D. degree from Zhejiang University, Hangzhou, China, in 1998, where he was a Lecturer with the Department of Biomedical Engineering, from 1998 to 2000, and became an Associate Professor in 2001. Since 2002, he has been with the Department of Computer Science and Engineering, Hangzhou Dianzi University, and was appointed as a Professor in 2009. He is currently a Professor with the School of Cyberspace, Hangzhou Dianzi University, Hangzhou, Zhejiang, China. He is also an Executive Director of the Information Security Laboratory. His current research interests include information system, model driven architecture, data mining, and digital heath.

Kim-Kwang Raymond Choo holds the Cloud Technology Endowed Professorship at the University of Texas at San Antonio (UTSA). He is the founding co-Editor-in-Chief of ACM Distributed Ledger Technologies: Research & Practice, and the founding Chair of IEEE TEMS TC on Blockchain and Distributed Ledger Technologies. He is also the recipient of the 2019 IEEE Technical Committee on Scalable Computing Award for Excellence in Scalable Computing (Middle Career Researcher), the British Computer Society's 2019 Wilkes Award Runner-up, the Fulbright Scholarship in 2009, the 2008 Australia Day Achievement Medallion, and the British Computer Society's Wilkes Award in 2008. He has also received best paper awards from IEEE Systems Journal in 2021, 2021 IEEE Conference on Dependable and Secure Computing (DSC 2021), IEEE Consumer Electronics Magazine for 2020, Journal of Network and Computer Applications for 2020, EURASIP Journal on Wireless Communications and Networking in 2019, IEEE TrustCom 2018, and ESORICS 2015.

Zheng Zhang received his M.S. degree in Computer Science from Zhejiang University, Hangzhou, China, in 2005. He was a visiting scholar at the University of Kentucky, USA, from 2013 to 2014. He has been working in Hangzhou Dianzi University as a faculty since 2000, where he is currently an Associate Professor in the university's School of Cyberspace. His current research interests include information security, information hiding, and confidential technology.

Gongxun Miao received the B.S. degree in Computer Science and Technology from Shandong Institute of management in 2012. He received the M.S. degree from Shandong University for Science& Technology in 2018. He is currently a member of the Standardization Technical Committee Information in Shandong Province and the Deputy Chief Engineer of Zhongfu Information Co., Ltd. He is mainly engaged in the technical research and development in the field of information security.

Yizhi Ren received his PhD in Computer software and theory from Dalian University of Technology, China in 2011. From 2008 to 2010, he was a research fellow at Kyushu University, Japan. He is currently a professor with School of Cyberspace, Hangzhou Dianzi University, China. His current research interests include network security, complex network, and trust management. Dr. Ren has published over 60 research refereed journal and conference papers. He is the recipient of IEEE Trustcom 2018 Best Paper Award, CSS 2009 Best Student Paper Award, and AINA 2011 Best Student Paper Award.