Computers & Security 120 (2022) 102831

Contents lists available at ScienceDirect

Computers & Security

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

XSS adversarial example attacks based on deep reinforcement learning

Li Chen, Cong Tang, Junjiang He, Hui Zhao*, Xiaolong Lan, Tao Li

School of Cyber Science and Engineering, Sichuan University, Chengdu, China

ARTICLE INFO

Article history:
Received 14 March 2022
Revised 16 June 2022
Accepted 8 July 2022
Available online 10 July 2022

Keywords:
Web security
Cross site scripting
Adversarial examples
Adversarial attack
SAC
Reinforcement learning

ABSTRACT

Cross-site scripting (XSS) attack is one of the most serious security problems in web applications. Although deep neural network (DNN) has been used in XSS attack detection and achieved unprecedented success, it is vulnerable to adversarial example attacks because its input-output mapping is quite discontinuous to a large extent. The existence of adversarial examples have raised concerns in applying deep learning to key security fields. Therefore, to evaluate the effectiveness of these detection methods, a XSS adversarial example attack technique using Soft Actor-Critic (SAC) reinforcement learning algorithm is presented in the paper. A key aspect of our idea is to train an agent using SAC algorithm to build adversarial examples for several popular XSS detection models which have been proved can achieve very high accuracy rate by simulation experiments. We first design mutation strategies for different modules of XSS attack vectors to ensure the validity of the generated adversarial examples. Then, the agent selects an appropriate escape strategy according to the feedback of the detection model until it bypasses the detection model. The final experiment results show that our model can achieve an escape rate of more than 92% and outperforms the latest method by up to 6%. In other words, the effectiveness of these detection models needs to be improved, at least in terms of defense adversarial example attacks.

1. Introduction

With the continuous development of the Internet, web applications have been greatly promoted, which brings great convenience to people's life, but also causes a series of security risks. Cross-site scripting (XSS), XML external entity injection (XXE), and cross-origin resource sharing (CORS) are common web attacks. Among them, attackers usually use XSS attacks to steal information and spread worms. So far, researchers have proposed many methods to defend against XSS attacks, such as code sanitization and content security policy (CSP), but they all have some limitations. For example, CSP does not allow the execution of inline JavaScript (JS) code (the biggest source of XSS attacks). However, in the actual development process, it is difficult to completely eliminate the inline scripts in the web page, which greatly limits the use of CSP. As of 2021, XSS attack is still one of the top ten web security threats provided by the Open Web Applications Security Project (OWASP).

At present, the research on XSS attack is mainly divided into two directions: XSS vulnerability discovery and XSS attack detection. Static analysis and dynamic analysis are often used for XSS vulnerability discovery. The static analysis explores potential attacks by analyzing the source code, but usually the source code is not public for security reasons (Doupe et al., 2013; Kronjee et al., 2018; Mohammadi et al., 2017; Steinhauser and Gauthier, 2016). The dynamic analysis finds attacks by simulating user's operations. It does not need source code, but it also has some limitations, such as a high false negative rate (because test cases can not cover all possible situations) (Fazzini et al., 2015; Lekies et al., 2013; Stock et al., 2014). Therefore, only relying on XSS vulnerability discovery is difficult to fully defend against XSS attacks. Some researchers try to use machine learning methods to detect XSS scripts (Likarish et al., 2009; Mereani and Howe, 2018; Nunan et al., 2012; Rathore et al., 2017; Wang et al., 2014). Although these machine learning methods have a high accuracy rate, they can only extract features manually. Deep learning has achieved remarkable success in the field of artificial intelligence (Andor et al., 2016; Hinton et al., 2012; Krizhevsky et al., 2012), which has inspired scholars to develop it to discover network attacks (Fang et al., 2018; Mokbal et al., 2019; Tekerek, 2021). The flexibility of DNN to small changes in data can generate a high-level invariant representation of the attack script, which enables the model to learn its nature (normal or malicious) even if it is a mutation script. The experiment results indicate that these deep learning models have better effects. However, the DNN model is vulnerable to adversarial example attacks, which has been proved in the case of different data types (Liang et al., 2018; Szegedy et al., 2014.) Adversarial examples exploit the highly nonlinear characteristics of the neural network to deceive the model

* Corresponding author.
E-mail address: zhaohui@scu.edu.cn (H. Zhao).

https://doi.org/10.1016/j.cose.2022.102831
0167-4048/

through some subtle changes to the original examples. Such a slight modification is completely indistinguishable to human eyes, but it allows GoogleLeNet to classify a panda image as a gibbon with 99.3% confidence (Goodfellow et al., 2014). Recently, some researchers pointed out that adversarial example attack is still an inevitable problem when deep learning models are applied to XSS attack detection (Fang et al., 2019; Zhang et al., 2020). So they proposed mining the adversarial examples of the detection model and training the detection model again with the obtained adversarial examples to improve the ability of the model to defense against adversarial attacks.

However, there are some challenges in building XSS adversarial examples. XSS scripts have fixed syntax rules. Only specific rules can be followed in the construction process, otherwise it will be invalid. The escape rate of adversarial examples obtained in the existing research is very low, and even coding misuse may occur. To improve this problem, we propose an XSS adversarial attack model based on the SAC algorithm (Haarnoja et al., 2018), which uses entropy regularization to increase the randomness of policy selection. Compared with the deterministic policy algorithm used in previous studies, the SAC algorithm (Haarnoja et al., 2018) can explore all possible optimal paths and find more types of adversarial examples. The major contributions of this paper are as below:

(1) Effective escape strategies: A series of mutation rules are designed according to various modules of XSS attack vectors, ensuring legitimacy of generated adversarial examples.
(2) Environment state representation: An environment state representation rule is proposed to assure the reinforcement learning agent can understand its escape stage and take the next action.
(3) XSS adversarial attack model: We propose a XSS adversarial attack model based on SAC algorithm, which constructs adversarial examples for several current popular XSS detection models.

The rest of the paper is organized as follows. The second section gives a brief overview of the background and related work. Section 3 introduces the content of deep reinforcement learning involved in this paper. Section 4 is concerned with the methodology used for this study. Section 5 presents some interesting evaluation results related to the assessment of the availability of our framework. Finally, the summary and prospect are discussed in Section 6.

2. Background and related work

In this section, we will analyse the background on XSS attacks and the related work of XSS attack defense technology and XSS adversarial attack technology.

2.1. Background

In an XSS attack, the attacker injects the designed malicious code into the HTML page, which is finally executed by the victim's browser. The reason is that some servers do not verify the security of the user's input, so attackers can easily inject some malicious code into web pages through normal input methods. When the victim accesses the tampered page on the target server through the browser, the malicious code can be successfully executed because the browser is trusted. Stealing sensitive information that can be used for identity recognition is the purpose of XSS attacks, such as cookies. After the attack is successfully implemented, the attacker can use the identity of a legitimate user for illegal operations.

According to different XSS script injection methods, XSS vulnerabilities can be simply divided into three categories: Stored XSS, Reflected XSS and DOM-based XSS. Stored XSS is also called persistent XSS. Malicious code is injected into the web page by an attacker and stored in the server's database. This means that all users accessing this page may execute this malicious script, so Stored XSS is more harmful. It is usually deployed in the comment area of a forum or a blog. Reflected XSS is called non-persistent XSS because it is one-time for visitors. Attacker usually induces victims to visit a URL containing malicious code through specific means (such as e-mail). When the victim clicks these specially designed URLs, they general inadvertently submit a specially made form, so that the injected malicious code reaches the target website and then returns to the victim's browser. This XSS usually appears in the search bar or login of a website to steal user cookies. DOM-based XSS is also called non-persistent XSS. The difference between it and the previous two types is that it does not require server participation. The attacker modifies the DOM structure of the web page through input and then triggers the attack through the DOM parsing of the client. Therefore, it may take a lot of analysis of the code flow to discover a DOM-based XSS attack. Although the utilization methods of the three XSS vulnerabilities are different, the attack examples for them are the same, so this paper can be used to generate adversarial examples of all vulnerability types.

2.2. XSS attack defense technique

According to different analysis methods, we divide the existing research into two categories: XSS vulnerability discovery and XSS attack detection. These two research directions have produced many new research results in recent years.

2.2.1. XSS vulnerability discovery

Static analysis and dynamic analysis are two mainstream methods for XSS vulnerability discovery. The static analysis discovers potential attacks by searching all possible execution paths in the code. The dynamic analysis tracks the data flow to find the injection point and then constructs attack vectors to verify whether there is a real vulnerability.

Steinhauser and Gauthier (2016) implemented a static detection tool (JSPChecker) that can analyze context-sensitive XSS vulnerabilities. JSPChecker judges XSS flaws according to the matching of sterilizer sequence and output context. Among them, sanitisation routines used on user's input were traced by data flow analysis. String analysis and fault-tolerant parsing are combined to simulate the output context of the sanitised value. Doupe et al. (2013) proposed separating the code and data in the web page and then combining them with CSP to defend against server-side XSS vulnerabilities. However, they can do nothing about the possible attacks in the JS code generated dynamically during the access process. Mohammadi et al. (2017) proposed an automated unit test method to find XSS vulnerabilities due to the wrong coding of input data. They built a set of test units for each JSP file and then designed a generator to automatically generate XSS attack strings. Finally, they confirm whether there are XSS attacks by executing the automatic test unit injected with an XSS attack string in JWebUnit. Kronjee et al. (2018) put forward a new static method to detect SQL injection and XSS vulnerabilities in PHP programs. They use static analysis to extract features from the source code, and then combined them with a machine learning classifier to detect vulnerabilities. However, they only got a 79% precision rate. Lekies et al. (2013) presented a method to detect and verify DOM-based XSS vulnerabilities. They track sensitive calls and sensitive attributes. Once they find that data flows into these sensitive calls, they try to attack with the prepared attack script. If successful, it indicates that there is a XSS vulnerability. Fazzini et al. (2015) proposed a technology to automatically add CSP to web applications. It determines which dynamic content can be loaded by the browser through stain tracking and then automatically modifies the server page to have such permissions.

2.2.2. XSS attack detection

The XSS vulnerability discovery methods are mainly to find the injection points through data flow analysis or stain tracking, and then construct an attack script to verify whether the vulnerability really exists. With further research, many XSS attack detection methods based on machine learning and deep learning have been proposed by researchers. It typically represents the examples as eigenvectors and models these vectors to judge whether they are malicious.

Because existing obfuscation technology makes detecting XSS more difficult, some researchers have offered certain approaches for dealing with obfuscated malicious code. Likarish et al. (2009) proposed using JS keywords and symbols as features to detect the confused malicious JS code, but they only achieved 92% accuracy. Nunan et al. (2012) improved Likarish's method, further refined the features, and achieved better detection results. Mereani and Howe (2018) extracted 38 structural features (symbols, like "<" and "?") and 21 behavioural features (HTML tags, etc) to identify confused XSS scripts. Their method achieves a high accuracy of 99.75%. Rathore et al. (2017) extracted three features from the webpages of social networks: URLs, webpages and SNSs, and then used the Random Forest algorithm to detect XSS attacks in social networks. They finally achieved an accuracy rate of 97.2% and a false positive rate of 8.7%.

Deep learning methods are used to make up for the shortcomings of traditional machine learning methods because they are capable of generating high-level feature representation after learning context information. Fang et al. (2018) put forward a fresh XSS attack detection method based on deep learning. They trained a Word2Vec (Mikolov et al., 2013) model to extract the features of the attack vector, and the extracted features were used to train the Long Short-Term Memory (LSTM) model. The experiment results showed that the precision and recall of their method are 99.5% and 97.9%. Mokbal et al. (2019) designed a new feature selection technology and a multilayer perceptron (MLP) model for detecting XSS, and then constructed a large dataset to train the model. The detection scheme has achieved excellent results on a newly used test dataset, and the accuracy and recall are 99.32% and 98.35%, respectively. Tekerek (2021) proposed detecting common web attacks using a Convolution Neural Network (CNN). The overall architecture consists of two parts: data preprocessing and CNN model. The proposed CNN deep learning algorithm can reach 97.07% accuracy on the CSIC2010v2 datasets. Despite the fact that the methods above can effectively detect XSS attacks, they do not consider the impact of adversarial examples. If an attacker constructs adversarial examples for a specific model, then the model may not work.

2.3. Adversarial attack technique

Deep learning has been widely used in image recognition, attack detection, and other prediction problems. Although deep learning methods significantly improve the accuracy of prediction, they are easily subjected to adversarial example attacks, which have attracted the attention of many researchers. Chen et al. (2020) designed an uncomplicated part substitution model which is broadly similar to the goal model of an automatic speech recognition system. For 98% of the commands of the target equipment (like Google Assistant and Microsoft Cortana), their method can generate at least one adversarial example to attack. Zhou et al. (2021) extracted the common weaknesses of face recognition models through ensemble learning, then successfully deceived the white box and black box face recognition systems of PC terminal and mobile terminal. Sun et al. (2020) proposed a general black-box adversarial attack method against the perception module in autonomous driving, which generally reaches about 80% mean success rates on all target models. Similarly, these XSS detection models will make wrong judgment on the adversarial examples carefully designed by the attacker. The existing studies about XSS adversarial attacks are as below:

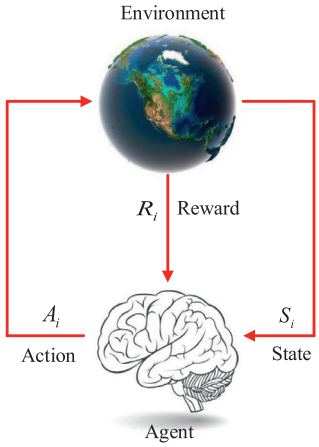

Fig. 1. Decision making process of reinforcement learning.

Fang et al. (2019) designed an XSS adversarial attack model based on the Dueling Deep Q Networks algorithm. Because the bypass strategy they chosen is relatively simple, the escape rate is not high, at only less than 10%. According to the Monte Carlo Tree Search (MCTS) algorithm, (Zhang et al., 2020) proposed an algorithm called MCTS-T to construct XSS adversarial examples, then train the detection model with the adversarial examples to improve the detection rate of the model. However, only a few simple escape strategies (like adding illegal characters) were used, and the time complexity was very high. Wang et al. (2022) put forward an XSS adversarial attack method using the soft Q-learning algorithm, which divided the whole bypass process into the HTML bypass stage and JavaScript bypass stage. Finally, their method achieves better performance, and the escape rate reaches about 85%.

The existing research has the problems of low escape rate and coding misuse. To overcome these shortcomings, we propose mutation rules for different modules of the XSS attack vector. Then, an intelligent agent is trained to generate adversarial examples according to these mutation rules to improve the escape rate.

3. Deep reinforcement learning

3.1. Problem definition

In this work, we focus on generating adversarial examples that can bypass XSS detection models. In the real scene, we usually can't know the details of the detection model. That is, we can only take black-box attack. If we have a strong knowledge base, this task may be completed. However, this is not a wise way but belongs to violent cracking. The number and type of the obtained adversarial examples completely depend on this knowledge base. In fact, human methods can be simulated by an intelligent reinforcement learning agent learning from experience, rather than people having to list all the possibilities and create all the solutions. Therefore, we chose to train a reinforcement learning agent to solve this problem. The agent will explore the optimal strategy in a given environment in the process of continuous learning.

Like supervised learning and unsupervised learning, reinforcement learning is a branch of machine learning. In reinforcement learning, the whole system consists of two subjects, as shown in Fig. 1, agent (Brain in Fig. 1) and environment (Earth in Fig. 1). The agent is in environment E, and the state space is X, where each state $x \in X$ is the description of the environment received

by the agent. The actions that agent can take from action space A. At a certain moment t, the agent selects an action $a_t \in A$ and acts on the current state s_t, which will make the environment transfer from the current state to a new state s_{t+1} according to a certain probability. At the same time, the environment will feed back a reward r_{t+1} to the agent according to the reward function R. The reward reflects the quality of the agent's decision-making. The agent's goal is to maximize long-term reward $R = \sum_{i=0}^{\infty} \gamma^t r$ where $\gamma \in [0, 1]$ is the reward discount factor applied to reduce the weight of follow-up rewards.

3.2. Types of reinforcement learning

Proximal Policy Optimization (PPO) (Schulman et al., 2017) algorithm is the most mainstream deep reinforcement learning algorithm at present. But PPO is an on-policy algorithm and requires a large number of sampling to learn, which is unacceptable for many real scenes. Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016) algorithm is an off-policy algorithm. It does not depend on a great quantity of examples for learning like PPO, but it is sensitive to super parameters and has poor convergence effect. SAC algorithm (Haarnoja et al., 2018) is an off-policy algorithm which uses entropy regularization to increase the randomness of policy selection. It has the following advantages:

- Stronger exploration ability. For the deterministic strategy algorithm, only one optimal action is considered in each state. However, the core idea of maximum entropy is to explore all possible optimal paths without leaving any useful action.
- Faster convergence. Increasing entropy leads to more exploration, which can speed up later learning. It also can prevent the policy from converging to the bad local optimum too early.
- Easier to expand. The algorithm can be easily extended to very complex high-dimensional tasks, in which no policy methods (such as DDPG (Lillicrap et al., 2016)) are typically difficult to obtain good results.

In most reinforcement learning algorithms, agent constantly interacts with the environment to search the most appropriate path. However, an important feature of SAC (Haarnoja et al., 2018) is entropy regularization. Increasing entropy means that the choice of strategy is more random. Therefore, it will explore all possible optimal paths, not just one solution. Thus, compared with the traditional reinforcement learning algorithm, SAC is more suitable for mining adversarial examples, because a detection model can not have only one kind of adversarial examples.

3.3. SAC algorithm

The learning objective of general reinforcement learning is very direct, which is to learn a strategy to maximize the expectation of accumulated reward:

$$\pi^* = arg\max_{\pi} \underset{\tau\sim\pi}{E}\left[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t))\right] \tag{1}$$

where γ is the reward discount factor. In addition to the above basic objective, maximum entropy reinforcement learning also requires that the entropy of each action output by the strategy is the maximum:

$$\pi^* = arg\max_{\pi} \underset{\tau\sim\pi}{E}\left[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t)) + \alpha H(\pi(\cdot|s_t))\right] \tag{2}$$

where α is the trade-off coefficient, $H()$ denotes entropy and $\pi(a|s_t)$ represents the probability of taking action a in state s_t.

Algorithm 1: Soft Actor-Critic.

```
Initialize parameters ψ, ξ, θ, φ
Clear experience pool D
for each iteration do
  for each environment step do
    Select action a_t based on Actor network
    Execute a_t to get status s_{t+1} and reward r(s_t, a_t)
    Store (s_t, a_t, r(s_t, a_t), s_{t+1}) in D
  end for
  for each gradient step do
    Update parameter ψ of V network
    Update parameter ξ of two Q network
    Update parameter θ of Actor network
    Update parameter φ of target V network
  end for
end for
```

Meanwhile, the action value function Q and the state value function V also need to be changed. Q function is defined as

$$Q(s_t, a_t) = \underset{\tau\sim\pi}{E}\left[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t)) + \alpha \sum_{t=1}^{\infty} \gamma^t H(\pi(\cdot|s_t))\right] \tag{3}$$

and V function is:

$$V(s_t) = \underset{\tau\sim\pi}{E}\left[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t)) + \alpha H(\pi(\cdot|s_t))\right] \tag{4}$$

In this paper, our model uses SAC (Haarnoja et al., 2018) to automatically mine XSS adversarial examples. When implementing SAC, all functions are fitted by neural networks. There are five neural networks, one Actor network, two V networks (V and target V) and two Q networks. Actor network is used for policy selection and several other networks provide information for Actor's decision-making. Both V function and Q function have two networks because the network needs to be updated. The realization of the algorithm is shown in Algorithm 1.

4. Proposed model

As shown in Fig. 2, the architecture of the method proposed in this paper consists of a detection phase and an escape phase. In the detection phase, for each XSS example, we first preprocess it and then input it into the detection models. If it is identified as malicious, the reward and state will be input into the adversarial model. For the escape phase, the agent chooses an appropriate escape action according to the current environment state and then updates parameters of the neural network based on the reward feedback provided by the XSS detection model. According to the escape strategy selected by the agent, the example is modified and then input into the detection model. This process is repeated until the detection model judges that the example is benign, so as to obtain an adversarial example.

4.1. XSS detection model

In order to evaluate the effectiveness of our method on XSS detection models, we established three XSS deep learning detection models, namely LSTM-based, MLP-based, and CNN-based. In addition, we also selected two mature XSS detection tools (XSSChop Chaitin (2019) and Safedog (2020)) for testing. They detect XSS attacks based on different methods. Three deep learning models are based on word embedding. SafeDog is based on rule matching, and XSSChop is based on semantic analysis.

4.1.1. Word2Vec

Word2Vec (Mikolov et al., 2013) uses a layer of a neural network to map the word vector in a one-hot representation to a

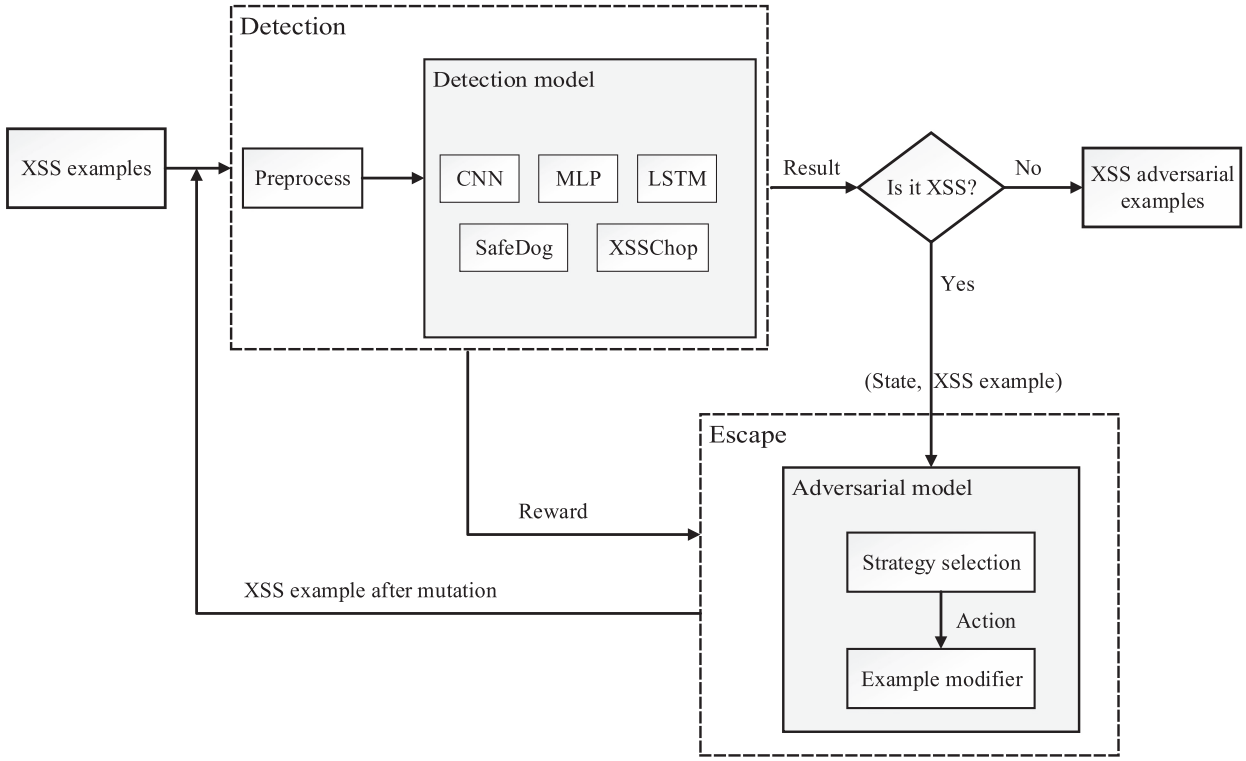

Fig. 2. The framework of adversarial attack model.

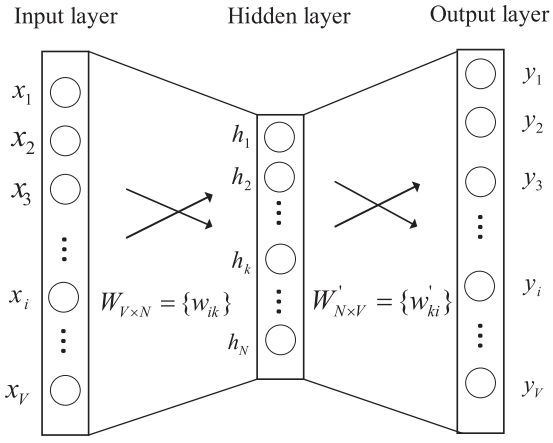

Fig. 3. A simple CBOW model structure.

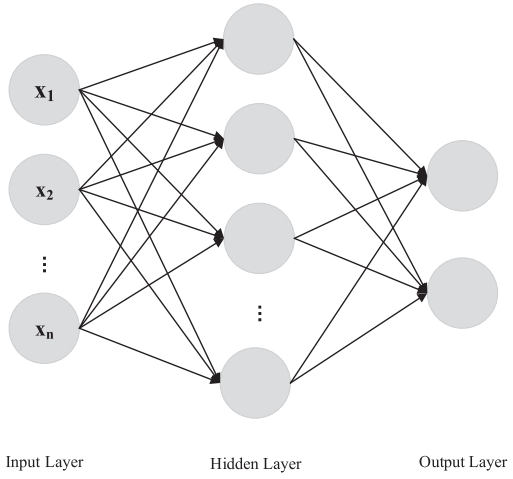

Fig. 4. Three layer MLP model structure.

distributed representation. It has two training modes: continuous bag-of-words (CBOW) and skip-gram. CBOW uses the context to predict the current value, while skip-gram uses the current value to predict the context. This paper chose the CBOW model because it has a faster training speed. Fig. 3 shows a simple CBOW model, including the input layer, hidden layer, and output layer. Where the input x is the one-hot representation of a word, and the output y is the probability of the V words (assuming a total of V words). The neural network is trained by back propagating the error between the output y and the real y. When the model is trained, the final result is actually the weight matrix W of the neural network. Because the position of 1 in the one-hot coding of each word is different, the vector obtained by multiplying W is different. This vector can be used to uniquely represent x, that is, the word vector we want. Generally, the dimension N of a word vector (equal to the number of nodes in the hidden layer) is much smaller than the total number of words V. The CBOW model usually inputs multiple words (using context to predict the current value), so Word2Vec can achieve dimension reduction while capturing context information.

4.1.2. Deep learning detection models

MLP is a forward-propagation artificial neural network (ANN). In addition to the input layer and output layer, it can have multiple hidden layers in the middle. The simplest MLP contains only one hidden layer, as shown in Fig. 4. MLP has strong learning ability and can be used to classify and model nonlinear data, so it is often used in attack detection research. The MLP model used in this paper contains two hidden layers, the number of neurons is

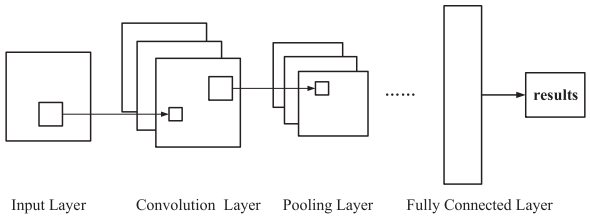

Fig. 5. CNN model structure.

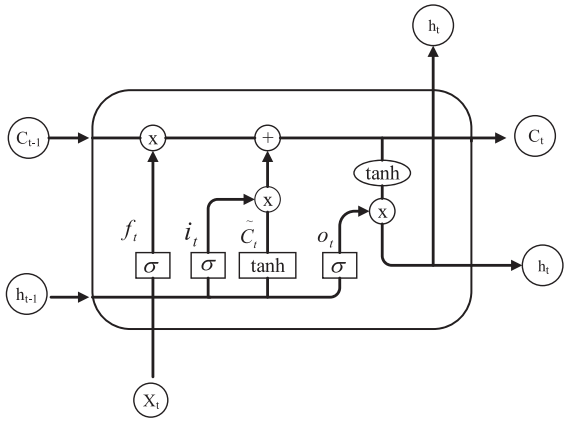

Fig. 6. LSTM model structure.

100 and 20 respectively, and the activation function is relu. Softmax function is selected in the output layer.

A CNN model usually includes convolution layers, pooling layers, and full connection layers. The convolution layer is composed of one or more convolution cores, which is used to extract the characteristics of the input. The pooling layer is used for the optimization calculation. It will delete redundant feature information and reduce the dimensions of features. The full connection layer learns the distribution of features in high-dimensional space and outputs the classification results to the output layer. Fig. 5 shows a simple CNN model structure. 1-CNN is the CNN model utilized in this paper. Two convolution layers with a filter size of 64 and a pooling layer with a pool size of 2 are added first, followed by two convolution layers with a filter size of 128 and a pooling layer with a pool size of 2, and lastly a full connection layer. The input dimension of this CNN model is 200.

LSTM (Graves, 2012) is a special recurrent neural network. It regards each neuron as a memory cell and controls the cell through three gates. Fig. 6 shows the structure of a memory cell of LSTM.

The forget gate (f_t in Fig. 6) determines what information is discarded from the previous cell. The gate will read the output result h_{t-1} of the previous cell and the input information x_t of the current cell, and output a value between 0 and 1 through the sigmoid function. $f_t = 1$ means to be completely retained and $f_t = 0$ means to be completely discarded. In Eq. (5), W_f and b_f represent the weight and offset of the current cell respectively.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{5}$$

The input gate determines what new information to add to the cell. This step includes two parts, as listed in Eq. (6) and Eq. (7). The sigmoid function determines which values will be updated. The tanh function outputs a new candidate cell $\tilde{C}_t$. Finally, the results of the two parts are multiplied to update the cell, as shown in Eq. (8).

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{6}$$

Table 1
Tokenization rules.

Tokenization	Example
HTML tags	<script>, </textarea>, <a
Attributes	href=, src=
Pseudo protocol	javascript:, data:
Attack code	alert(
URL	http://
Content	'document.cookie'
Special symbols	&, #, /,), >

$$\tilde{C}_t = tanh(W_c[h_{t-1}, x_t] + b_c) \tag{7}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{8}$$

The output gate outputs the final state of the current cell, as shown in Eq. (9) and Eq. (10).

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o] \tag{9}$$

$$h_t = o_t * tanh(C_t) \tag{10}$$

The LSTM model used in this paper contains 128 neurons and the classification function of the output layer is softmax function.

4.1.3. SafeDog and XSSChop

Safedog (2020) is a popular commercial web application firewall (WAF) in China. It judges the legitimacy of the data by matching the rules in its feature library. This paper uses it as a detection model based on rule matching for attack testing. XSSChop (Chaitin, 2019) is a WAF product based on intelligent semantic analysis launched by a Chinese network security company. It contains four steps: encoding processing, lexical extraction, syntax detection, and semantic judgment. On the basis of successful syntax parsing, it judges the degree of harm and finally gives the risk level. Compared with the detector based on rule matching, it does not need to maintain the rule library and can deal with obfuscated examples. This paper uses it as a detection model based on semantic analysis for attack testing.

4.1.4. Detection method

As shown in Fig. 7, for each XSS example, the data processing process consists of three steps, preprocessing, tokenization, and word vector representation. After processing, the data will be input into the detection model. The detailed process is as follows:

Preprocessing: First, because the positive examples in the training set are obfuscated malicious examples, the de-obfuscated operations need to be completed before the unified processing of the examples, such as decoding. However, we can only do some simple operations to remove obfuscation. It is impossible to completely remove obfuscation because the obfuscation rules are complex and changeable in reality. Secondly, in order to reduce the interference of redundant information, we need to normalize the examples. For example, we uniformly convert the examples to lowercase, replace the URL with "http://", and remove special characters like "<br>".

Tokenization: Text classification first requires tokenizing the text. Combining HTML syntax rules and humans' habit of recognizing XSS attack examples, the tokenization rules we designed are shown in Table 1. Each example is decomposed into a group of words after tokenization, as shown in the second step in Fig. 7.

Word vector representation: The data set we use is the effective XSS attack traffic collected by XSSed (KF, 2012) in the real network environment. In order to avoid detection, attackers usually add a large number of meaningless interference characters to the attack examples, so most of the words we get are meaningless words. Therefore, after experimental comparison (comparing the

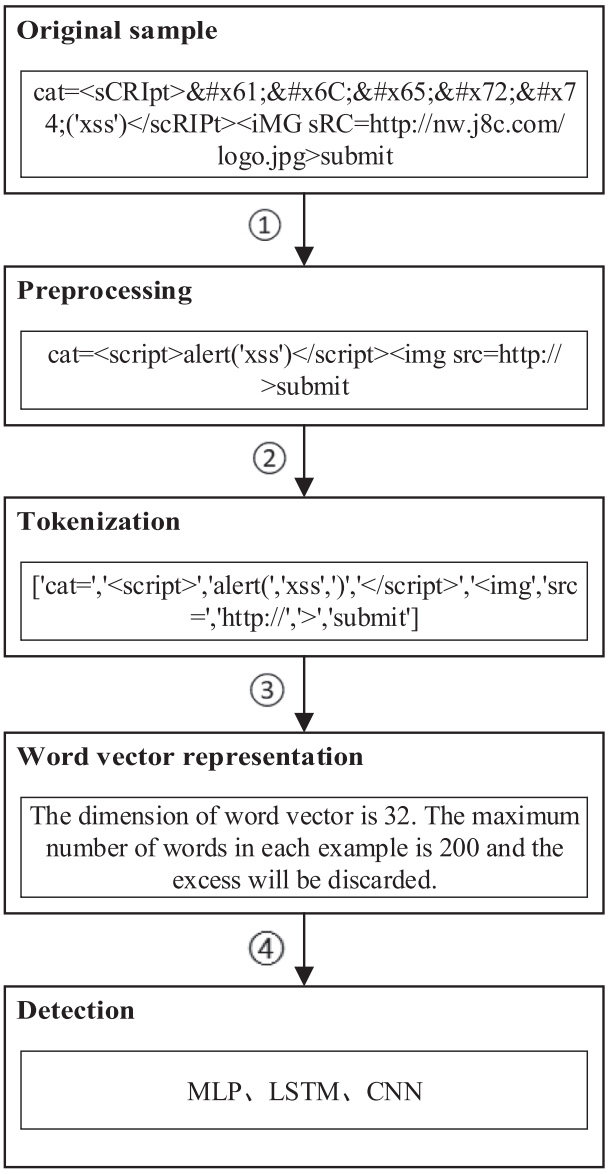

Fig. 7. XSS attack detection process.

effects of 2000, 3000, and 4000 words), we finally selected 3000 words (10% of the total) with the highest frequency in the XSS attack examples. In fact, these 3000 words have already covered the typical characteristics of XSS attack examples. All other words are replaced with "None", so the example in step 3 in Fig. 7 will become ['None', '<script>', 'alert(', 'None', ')', '</script>', '<img', 'src=', 'http://', '>', 'None'] and then be used to train the Word2Vec (Mikolov et al., 2013) model. Finally, each word is represented as a 32-dimensional word vector. Each example can have a maximum of 200 words. The extra words will be discarded, and less than 200 words will be filled with 0.

Detection: After preprocessing, each XSS example will be represented by a set of word vectors, and then the word vectors and example label are input into MLP, LSTM, and CNN detection model. For SafeDog and XSSChop, we don't need to deal with the original examples but directly call their interfaces for detection.

4.2. XSS adversarial attack model

This part describes the proposed adversarial model, that is, to formalize the reinforcement learning problem (as listed in the second step in Fig. 8). First, we define the actions that agent can take. Secondly, we design the change rules of the agent's state. Finally, we also define a reward function to reward the agent's action. After all this is done, the agent can be trained to mine adversarial examples. The agent's state is sent into the neural network that produces the policy outputs, from which the agent selects an action by sampling. Then, the agent's state is renewed as described in Section 4.2.2. The example modified based on this action is input into the detection model, and then the reward and result fed back to the agent are utilized to update the neural network. This process will continue till the agent successfully escapes the detection model or the maximum quantity of steps (described in Section 5.3.2) in a round is reached.

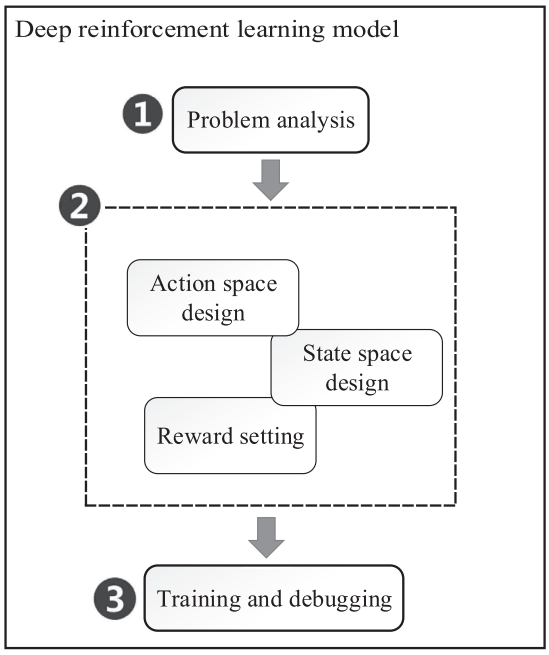

Fig. 8. Flow chart of solving problem with reinforcement learning.

4.2.1. Action space

We define the modification operations actually implemented in the XSS attack vector as agent actions. Table 2 lists some common XSS attack vectors. Actually, there are many types of XSS attack vectors, and they all follow certain syntax rules and characteristics. XSS attack vectors generally include the following parts:

(1) HTML tagsHTML tag is the most basic unit in HTML. Tags that can contain special attributes (like src) can become components of XSS attack vectors. The commonly used tags in XSS attack vectors are <a>, <img>, <script>, and so on.
(2) Special attributesBecause the XSS attack code must be executed in the JavaScript environment, special attributes such as 'src' and 'href' can trigger the browser's JavaScript parser to complete the parsing of XSS attack code, so as to implement attack.
(3) Pseudo protocolIn HTML, special attributes cannot directly load attack code, which can be completed with the help of pseudo protocol. Commonly used pseudo protocols are "javascript:", "data:".
(4) EventsEvents also belong to HTML attributes. Unlike other attributes, JavaScript attack code can be directly added after events without pseudo protocol to specify the code type. However, the event itself needs specific conditions to trigger, such as mouse click and drag.

Table 2
Common XSS attack vectors.

Number	XSS attack vector example
1	<img src=# onerror=prompt(8)>
2	txtsearch=’><script>alert(1)</script>
3	question=</textarea><iframe src=javascript:alert(1); />
4	itemid=’ onmouseover=alert(document.cookie) bad=’
5	<body style = "background-image:url(javascript:alert(1))" > </body>
6	data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K

Table 3
Actions.

A1.Add " &#14" before "javascript"	A15. Replace "(" and ")" with grave note
A2. Mixed case HTML attributes	A16. Encode data protocol with Base64
A3. Replace spaces with "/", "%0A" or "%0D"	A17. Remove the quotation marks
A4. Mixed case HTML tags	A18. Unicode encoding for JS code
A5. Remove the closing symbol of the single tags	A19. HTML entity encoding for "javascript"
A6. Add "
" to "javascript"	A20. Replace ">" of single label with "<"
A7. Add "&#x09" to "javascript"	A21. Replace "alert" with "top['al'+'ert'](1)"
A8. HTML entity encoding for JS code (hexadecimal)	A22. Replace "alert" with "top[8680439..toString(30)](1)"
A9. Double write HTML tags	A23. Add interference string before the example
A10. Replace "http://" with "//"	A24. Add comment into tags
A11. HTML entity encoding for JS code (decimal)	A25. "vbscript" replaces "javascript"
A12. Add ":" to "javascript"	A26. Inject empty byte "%00" into tags
A13. Add " " to "javascript"	A27. Replace "alert" with "top[/al/.source+/ert/.source](1)"
A14. Add string "/drfv/" after the script tag	

(5) Attack codeAttack code is the core of XSS attack vector. It has many construction methods according to different attack purposes. Taking pop-up class as an example, there are three pop-up methods in JavaScript: alert(), confirm() and prompt().
(6) Special symbolsSpecial symbols generally include "<", ">", "(", ")", "/", and single and double quotation marks. The symbols "<", ">", and "/" are used to introduce HTML tags. "(" and ")" usually appears in attack code. And single and double quotation marks may appear in the attribute value.

The mutation rules applied to each module of the XSS attack vector are almost different. The attack vector will be rendered invalid if mutation rules are misused. Therefore, in order to avoid this, we propose to design mutation rules for each module of the XSS attack vector separately. Table 3 summarizes all the actions that the agent can perform. For action A23, we collected about 20000 strings, which are normal HTTP request parameters and come from Alexa (Cooper, 2020) top 100 websites by depth crawling. To create confusion, these strings are inserted into some positions of the attack vector at random. The action space is determined by the mutation rules we designed. Our principle of building action space is to make the action as concise as possible. That is, an action only represents modifying a module of an XSS attack vector. As a result, we can correlate the feedback provided by the detection model with the operation performed on the attack vector. As shown in Table 3, the final action space consists of 27 actions.

4.2.2. State space

We update the state of the agent by keeping the information that is relevant to the action taken by the agent. Each position of the state vector represents the action selected by the agent to mutate the attack vector in this step. Fig. 9 shows the state change process in an adversarial attack. At first, the state of the agent is $S_0 = [0, 0, 0, \cdots, 0]$, which means that there is no mutation on the attack vector, that is, original attack vector. Then, the attack vector is input into the detection model and identified as malicious. The agent selects action 4 (case mixing), so the state changes to $S_1 = [4, 0, 0, \cdots, 0]$. If there is no module related with the selected ac-

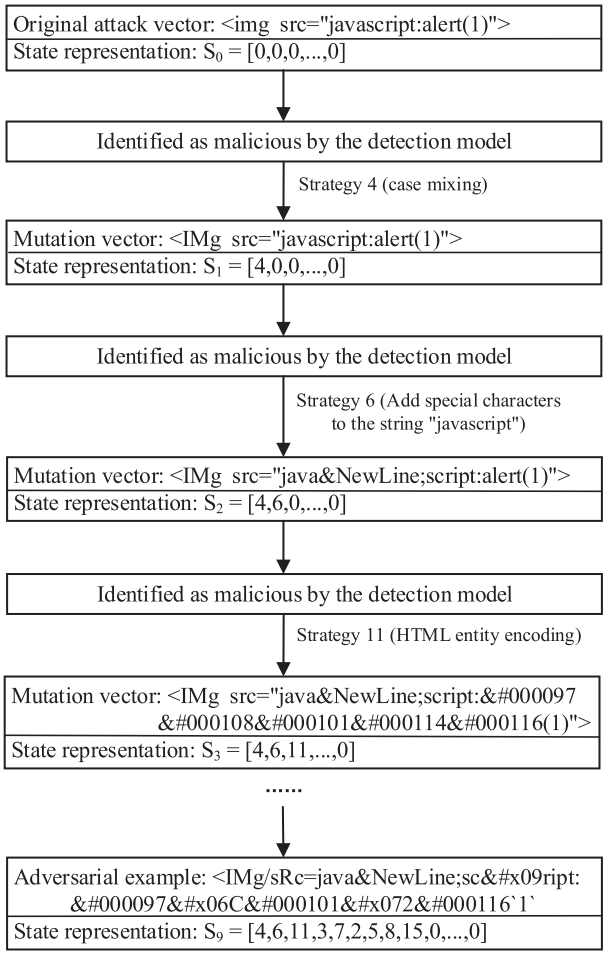

Fig. 9. The agent state update process.

tion in the attack vector, agent will reselect action. After case conversion of the corresponding module of the attack vector, the vector will be input into the detection model again. After being judged as malicious again, the agent selects action 6 (adding special string to JavaScript protocol) and mutates the attack vector accordingly, and then the agent's state changes to $S_2 = [4, 6, 0, \cdots, 0]$. This process will continue until the detection model determines that the attack vector is benign. The final attack vector is the adversarial example we seek. In this example, the final state of the agent is $S_8 = [4, 6, 11, 3, 7, 2, 5, 8, 15, 0, \cdots, 0]$, indicating that the detection model was bypassed by taking actions 4, 6, 11, 3, 7, 2, 5, 8 and 15 in turn.

4.2.3. Reward setting

Using rewards to formalize a goal is one of the most significant characteristics of reinforcement learning. In reinforcement learning, an agent always learns how to maximize rewards. The way we provide rewards must enable agent to maximize rewards and achieve our goals at the same time. Therefore, rewards can only be used to convey what you want to achieve, not how to achieve this goal, so rewards should not be set too carefully. After a series of experimental comparisons, we chose the simplest and most effective reward construction without any reward shaping, as shown in Eq. (11). The agent is given a reward of r=10 if the XSS example escapes the detection model successfully. Otherwise, it will be rewarded with -1. The experiment results show that this simple reward rule is enough to teach the agent to bypass the XSS detection model with as few actions as possible. The reason is that the reward will gradually decrease as the agent takes more action.

$$r = \begin{cases} 10, & successfully\ bypass\ detection\ model. \\ -1, & otherwise \end{cases} \tag{11}$$

5. Experiments

5.1. Dataset and parameter setting

Public datasets in the security field are very scarce. The experiment data used in this paper contains two parts. One is used to train the XSS detection models, while the other used for training the adversarial model.

The training set for XSS detection models contains around 90,000 examples. In particular, they consist of 40,637 malicious and about 50,000 benign examples. These examples were obtained separately. The malicious examples we used are the effective XSS attack traffic collected by XSSed (KF, 2012) in the real network environment. All benign examples are parameters of normal HTTP GET request, which come from Alexa (Cooper, 2020) top 100 websites by depth of crawling. To guarantee the comparability of the experiment results, we chose the same dataset as (Wang et al., 2022) for training the adversarial model, which is collected from the PortSwigger XSS cheat sheet (Portswigger, 2020). It contains almost all HTML tags and JavaScript trigger events, making it very comprehensive.

The detail experiment environment is listed in Table 4.

Table 4
Experiment environment.

Systerm	Ubuntu 20.04.3 LTS
CPU	CPU E5-2680 V4 @2.40GHz
GPU	NVIDIA GeForce RTX 3090 23G
Python module	Python 3.6.13 gensim=4.1.2 h5py=2.10.0 torch=1.10.0 keras=2.2.5 trnsorflow=1.14.0 gym=0.21.0 requests=2.26.0 nltk=3.6.5

5.2. Performance indicators

Evaluation indexes are needed to objectively evaluate the proposed mothod in academic research. We used escape ratio (ER) and detection ratio (DR) as evaluation indexes of the adversarial model. In addition, the commonly used indicators, including Accuracy, Precision, Recall and F1-Score, are chose as the evaluation indicators of the detection model.

True Positive (TP): indicates that XSS attack examples are correctly identified as malicious.

False Positive (FP): indicates that benign examples are wrong classified as malicious.

True Negative (TN): indicates that benign examples are correctly classified as benign.

False Negative (FN): indicates that XSS attack examples are wrong identified as benign.

Accuracy: refers to the ratio of the sum of malicious examples and benign examples correctly predicted to the total number of examples. Its definition is shown in Eq. (12).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

Precision: refers to the ratio of the number of correctly predicted malicious examples to all predicted malicious examples, which directly shows the performance of the model. It is defined as follows.

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

Recall: refers to the ratio of the number of malicious examples correctly predicted to all malicious examples, which is directly proportional to the quality of the model. It is defined as in Eq. (14).

$$Recall = \frac{TP}{TP + FN} \tag{14}$$

F1-Score: We hope that Precision and Recall are both high, but in fact, these two indicators are contradictory. F1 score considers both Precision and Recall, so that they can reach the highest and achieve a balance at the same time. Its definition is as follows.

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \tag{15}$$

DR: indicates the ratio of malicious examples processed by the adversarial model are still recognized as malicious by the XSS detection model. DR is positively correlated with the model's ability to defend adversarial examples attack and it is defined as below.

$$DR = \frac{Number\ of\ malicious\ examples\ detected}{Total\ number\ of\ adversarial\ examples} \tag{16}$$

ER: refers to the ratio of malicious examples processed by the adversarial model are recognized as benign by the XSS detection model. It is defined as Eq. (17).

$$ER = \frac{Number\ of\ benign\ examples\ detected}{Total\ number\ of\ adversarial\ examples} \tag{17}$$

5.3. Performance comparison

Our purpose is to generate adversarial examples for XSS detection models. As a result, the detection model also plays a critical role. The experiments in this paper are arranged as follows: Firstly, we train the XSS detection models to accurately detect XSS attacks. Then the XSS adversarial attack model is trained to build adversarial examples to test the effectiveness of the detection models when facing adversarial example attacks.

Table 5
Performance of the XSS detection models.

Classifier	Precision	Recall	Accuracy	F1
MLP	99.92%	98.00%	99.61%	98.96%
LSTM	99.76%	98.35%	99.65%	99.06%
CNN	99.85%	98.90%	99.76%	99.38%
XSSChop	99.61%	98.25%	99.14%	98.93%
SafeDog	100.00%	96.16%	98.47%	98.05%

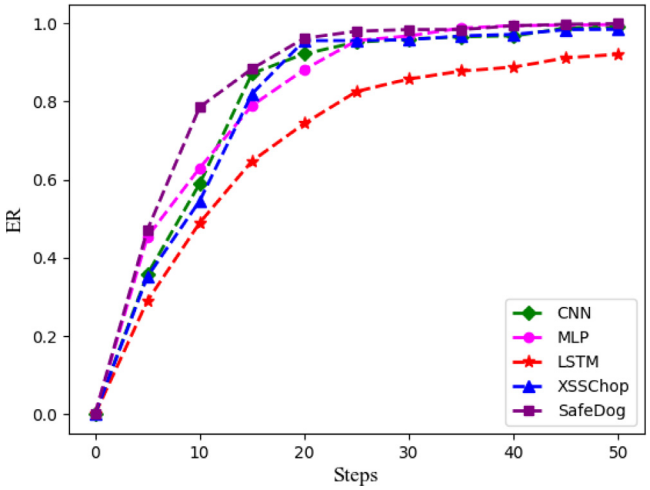

Fig. 10. Relationship between ER and the number of actions taken.

5.3.1. Detection model results

To examine the generated adversarial example's effectiveness, we selected five popular XSS detection models, three of which are deep learning models, and two are detection tools. The Precision, Recall, Accuracy and F1-Score of these models under the same dataset are shown in Table 5. From the data on the table, each classifier has good performance in detecting XSS attacks. The Precision of the MLP detection model was 99.92%; LSTM was 99.76%; CNN was 99.85%; XSSChop was 99.61%; and SafeDog was 100.00%. In terms of Recall and Accuracy, SafeDog was slightly lower than the other four classifiers, reaching 96.16% and 98.47%, but the other four classifiers were more than 98% and 99%. Moreover, the F1-Score of all classifiers exceeded 98%. These results verify the effectiveness of our models in detecting XSS attacks. In other words, these XSS detection models are able to accurately discover XSS attacks when they are not faced with adversarial example attacks.

5.3.2. Adversarial model setting

Because XSS scripts have fixed syntax rules, there is a limit to the position and number of times they can change. Otherwise their original semantics will be destroyed. In order to obtain a excellent adversarial model, this paper adjusts the parameter of step size in the reinforcement learning model. The step size refers to the number of actions the agent can select. We set the step size to different values to train the adversarial model, and calculate the ER of the five detection models under each step size. ER under different step values is calculated in Fig. 10. We can intuitively see that ER increases rapidly with the increase of step size at the beginning, then gradually tends to be stable. In CNN, MLP, XSSCop, and SafeDog models, the ER tends to be stable after the step size is 25, while in the LSTM model, the step size starts to be stable when it is close to 50. Finally, considering the trend of all models, we decided to set the step size to 50 so that all models can achieve the best performance.

Table 6
Adversarial results of detection models.

Detection model	Our method		Wang	
	DR	ER	DR	ER
XSSChop	1.54%	98.46%	5.80%	94.20%
SafeDog	0.05%	99.95%	3.10%	96.90%
LSTM	7.96%	92.04%	13.80%	86.20%
MLP	0.27%	99.73%	4.70%	95.30%
CNN	0.76%	99.24%	-	-

5.3.3. Adversarial model results

We input the original XSS attack examples into the adversarial model. Then the adversarial model generates adversarial examples for the five detection models separately after training. We chose the same dataset as (Wang et al., 2022) and did a comparative experiment. Compared with them, we have made two improvements. First, we proposed some more effective mutation strategies for different modules of XSS examples. Secondly, we chose to use the SAC algorithm (Haarnoja et al., 2018) to avoid the complexity and potential instability of the soft Q-learning algorithm. The final experiment results are shown in Table 6, where DR represents the detection rate and ER represents the escape rate.

From the results in the table, the ER of the adversarial examples generated in this paper on XSSChop was 98.46%; SafeDog was 99.95%; LSTM was 92.04%; MLP was 99.73%; and CNN was 99.24%. The comparison results indicate that our adversarial model obtains better performance compared to Wang (Zhang et al., 2020) in all detection models. Among them, we improved the ER by 4.26% on XSSChop, 3.05% on SafeDog, 5.84% on the LSTM model, and 4.43% on the MLP model. Except that the ER of the LSTM model was slightly lower, the ERs of the other four models were all over 98.00%. In other words, the existing popular XSS detection models perform poorly when facing adversarial example attacks.

5.3.4. Discussion

We will analyze which strategies are more effective for the three types of XSS detection models. Fig. 11 lists some adversarial examples. For the model based on rule matching, it is easy to bypass it by selecting some unusual tags or adding some new elements from HTML5. Because web front-end technology is developing rapidly and XSS syntax is very flexible. In this case, the typical rule-based models' knowledge base is insufficient to cover all attack modes, which makes it easy to cause false negative. Even the new HTML5 tag "<video>" can fool SafeDog in Fig. 11(2). For the model based on semantic analysis, adding some abnormal symbols such as "<" and "=" to the example can interfere with the model's analysis process of HTML syntax. Meanwhile, mixed coding may destroy the model's process of extracting JavaScript syntax. In Fig. 11(1), the agent added special characters to the malicious example and then performed mixed coding on the example (Unicode coding and HTML entity coding). For the deep learning detection model based on word embedding, adding some interference strings or complex substitution can effectively affect the feature extraction of the model. In Fig. 11(3) and (4), the agent adds some interference strings to the malicious examples and makes a complex replacement for the key code "alert(1)", which greatly reduces the detection rate of the model. For all adversarial examples generated, we can test their availability on Yahoo Webseclab platform Yahoo). It contains 31 XSS vulnerability test cases, covering most XSS attack scenarios. So it perfectly fits the context requirements for testing our examples.

Our work demonstrates that the existing XSS detection models are not functional against adversarial attacks. Our model can achieve such a high escape rate not only because we adopted more effective bypass strategies, but also because the dataset we chose

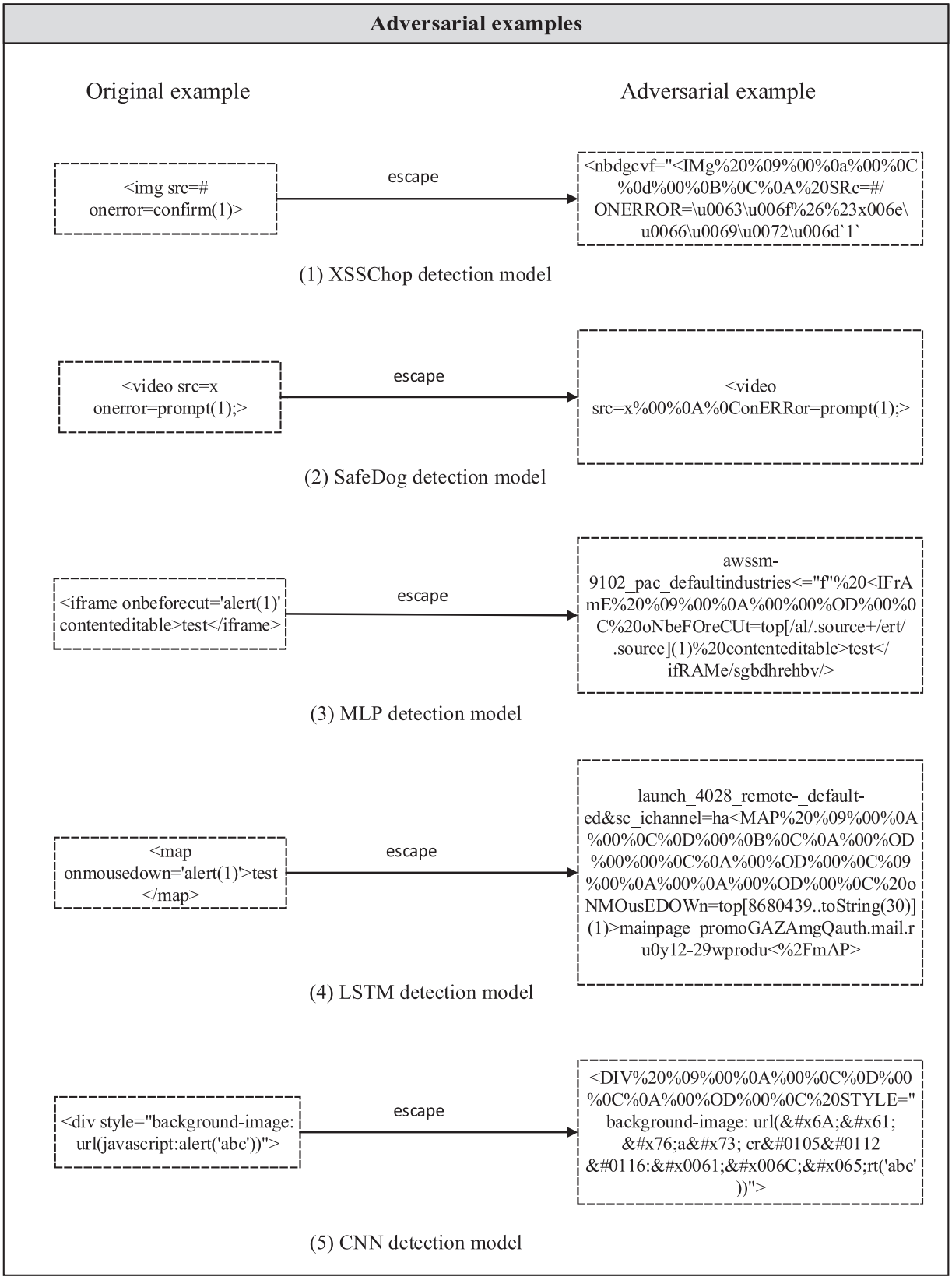

Fig. 11. Adversarial examples.

to generate adversarial examples is not exactly the same as the dataset of the detection model used for training. Of course, the difference between the two datasets will not have any impact on the agent learning bypass strategy. Actually, in the real scene, the information held by the attacker and the defender is unequal. The dataset we selected contains as many HTML tags and events as possible, but some tags and events may not exist in the training set of the detection model, so the escape rate can be improved to a certain extent.

5.4. Suggestions

In order to improve the ability of the model to detect adversarial examples, we should consider several aspects when building the detection model. First, XSS attack vectors can use multiple types of coding, so they should be decoded before extracting features. Second, the training set should cover as many tags and events as possible. You can manually construct some payloads to add to the training to ensure the training set is more comprehensive. Finally, there are a large number of HTML tags and events, so it is best to combine automatic feature extraction with manual feature extraction, otherwise it is easy to produce false negative or false positive.

6. Conclusion

This study seeks to explore the effectiveness of XSS detection models. For this purpose, we provide an XSS adversarial attack model based on the Soft Actor-Critic algorithm. In order to improve the escape rate, we design corresponding mutation strategies for different modules of the XSS attack vectors, and then train a reinforcement learning agent to select strategies, so as to make the detector output wrong results. The experiment results showed that our method can achieve an escape rate of over 92%. This also shows that the detection models based on deep learning can effectively detect XSS attacks, but it cannot resist adversarial attacks. Therefore, the key to ensuring the security of machine learning and deep learning in various applications is to further study the adversarial attack technology and put forward more useful defense methods.

Declaration of Competing Interest

None.

CRediT authorship contribution statement

Li Chen: Methodology, Writing – original draft. **Cong Tang:** Conceptualization, Writing – review & editing. **Junjiang He:** Conceptualization, Writing – review & editing. **Hui Zhao:** Software, Conceptualization, Writing – review & editing. **Xiaolong Lan:** Validation, Writing – review & editing. **Tao Li:** Data curation, Writing – review & editing.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China (No. 2020YF B1805400); in part by the National Natural Science Foun dation of China (No. U19A2068, No. U1736212, No. 62032002, and No. 62101358); in part by the China Postdoctoral Science Foundation (No.2020M683345); Fundamental Research Funds for the Central Universities (Grant No. SCU2021D052).

References

Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S., Collins, M., 2016. Globally normalized transition-based neural networks. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 2442–2452.

Chaitin, 2019. XSSChop: XSS detection engine. https://xsschop.chaitin.cn/.

Chen, Y., Yuan, X., Zhang, J., Zhao, Y., Zhang, S., Chen, K., Wang, X., 2020. Devils whisper: A general approach for physical adversarial attacks against commercial black-box speech recognition devices. In: 29th {USENIX} Security Symposium ({USENIX} Security 20), pp. 2667–2684.

Cooper, 2020. Alexa. https://www.alexa.com/.

Doupe, A., Cui, W., Jakubowski, M.H., Peinado, M., Kruegel, C., Vigna, G., 2013. deDacota: toward preventing server-side XSS via automatic code and data separation. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp. 1205–1216.

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.

Fazzini, M., Saxena, P., Orso, A., 2015. AutoCSP: Automatically retrofitting CSP to web applications. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1. IEEE, pp. 336–346.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets. Advances in neural information processing systems 27.

Graves, A., 2012. Long short-term memory. Supervised sequence labelling with recurrent neural networks 37–45.

Haarnoja, T., Zhou, A., Abbeel, P., Levine, S., 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International conference on machine learning. PMLR, pp. 1861–1870.

Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., et al., 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal processing magazine 29 (6), 82–97.

KF, 2012. XSSed: XSS attacks information. http://www.xssed.com/archive.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems 25.

Kronjee, J., Hommersom, A., Vranken, H., 2018. Discovering software vulnerabilities using data-flow analysis and machine learning. In: Proceedings of the 13th international conference on availability, reliability and security, pp. 1–10.

Lekies, S., Stock, B., Johns, M., 2013. 25 million flows later: large-scale detection of DOM-based XSS. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp. 1193–1204.

Liang, B., Li, H., Su, M., Bian, P., Li, X., Shi, W., 2018. Deep text classification can be fooled. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence, pp. 4208–4215.

Likarish, P., Jung, E., Jo, I., 2009. Obfuscated malicious javascript detection using classification techniques. In: 2009 4th International Conference on Malicious and Unwanted Software (MALWARE). IEEE, pp. 47–54.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2016. Continuous control with deep reinforcement learning. ICLR (Poster).

Mereani, F.A., Howe, J.M., 2018. Detecting cross-site scripting attacks using machine learning. In: International Conference on Advanced Machine Learning Technologies and Applications. Springer, pp. 200–210.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J., 2013. Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems 26.

Mohammadi, M., Chu, B., Lipford, H.R., 2017. Detecting cross-site scripting vulnerabilities through automated unit testing. In: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS). IEEE, pp. 364–373.

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.

Nunan, A.E., Souto, E., Dos Santos, E.M., Feitosa, E., 2012. Automatic classification of cross-site scripting in web pages using document-based and URL-based features. In: 2012 IEEE symposium on computers and communications (ISCC). IEEE, pp. 000702–000707.

OWASP. OWASP Top 10 2021. https://owasp.org/www-project-top-ten/.

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. Journal of Information Processing Systems 13 (4), 1014–1028.

Safedog, 2020. Safedog: web attack detection engine. http://www.safedog.cn/.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms.

Steinhauser, A., Gauthier, F., 2016. JSPChecker: static detection of context-sensitive cross-site scripting flaws in legacy web applications. In: Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, pp. 57–68.

Stock, B., Lekies, S., Mueller, T., Spiegel, P., Johns, M., 2014. Precise client-side protection against {DOM-based}{Cross-Site} scripting. In: 23rd USENIX Security Symposium (USENIX Security 14), pp. 655–670.

Sun, J., Cao, Y., Chen, Q.A., Mao, Z.M., 2020. Towards robust lidar-based perception in autonomous driving: General black-box adversarial sensor attack and countermeasures. In: 29th {USENIX} Security Symposium ({USENIX} Security 20), pp. 877–894.
Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R., 2014. Intriguing properties of neural networks. In: 2nd International Conference on Learning Representations, ICLR 2014.
Tekerek, A., 2021. A novel architecture for web-based attack detection using convolutional neural network. Computers & Security 100, 102096.
Wang, Q., Yang, H., Wu, G., Choo, K.-K.R., Zhang, Z., Miao, G., Ren, Y., 2022. Black-box adversarial attacks on XSS attack detection model. Computers & Security 113, 102554.
Wang, R., Jia, X., Li, Q., Zhang, S., 2014. Machine learning based cross-site scripting detection in online social network. In: 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS). IEEE, pp. 823–826.
Yahoo, Webseclab. https://github.com/yahoo/webseclab.
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.
Zhou, C., Jing, H., He, X., Wang, L., Chen, K., Ma, D., 2021. Disappeared Face: A Physical Adversarial Attack Method on Black-Box Face Detection Models. In: International Conference on Information and Communications Security. Springer, pp. 119–135.

Li Chen received her B.S. degree in Chongqing University of Posts and Telecommunications, Chongqing, Chi-na, 2020. She is currently pursuing the M.A. degree with the School of Cyber Science and Engineering, Sichuan University, Chengdu, China. Her current research interests include web security and artificial intelligence.

Cong Tang received his B.S. degree in Southwest Petr-oleum University, Chengdu, China, 2020. He is currently pursuing the M.A. degree with the School of Cyber Science and Engineering, Sichuan University, Chengdu, China. His current research interests include intrusion detection systems, network security and artificial intelligence.

Junjiang He received his B.S, M.S. degree in Southwest Petroleum University, Chengdu, Sichuan, China, in 2015 and 2018, respectively, and the Ph.D. degree in Sichuan University, Chengdu, Sichuan, China, in 2021. He is currently an Assistant Research Fellow in the School of Cyber Science and Engineering, Sichuan University, Chengdu, China. His current research interests include artificial immune system, cyber-physical system security, information security, cloud storage and data mining.

Hui Zhao received the B. S., M.S., the Ph.D. degree in Sichuan University, China, in 2000, 2003 and 2011, respectively. From 2007 to 2008, he was a visiting scholar in the University of Pittsburgh, PA, USA. He is currently an Associate Professor in the School of Cyber Science and Engineering, Sichuan University, Chengdu, China. His current research interests include network and information security, as well as the OS security.

Xiaolong Lan received the B. S. degree in mathematics and applied mathematics from Chengdu University of Technology and the M.S., Ph.D. degree in information and communication engineering from Southwest Jiaotong University, China, in 2012 and 2019, respectively. From 2017 to 2019, he was a visiting Ph.D. student with the University of Victoria, BC, Canada. He is currently an Associate Researcher in the School of Cyber Science and Engineering, Sichuan University, Chengdu, China. His current research interests include physical layer security, buffer-aided communication, energy-harvesting wireless communication, and mobile edge computing.

Tao Li received his Ph.D. degree in computer science from the University of Electronic Science and Technology of China, in 1994. He is currently a Professor in the School of Cyber Science and Engineering, Sichuan University, China. He is the Chief Scientist of the National Key Research and Development Plan for Cyberspace Security. He is also an editorial board member of Immune Computation and several other international academic journals. His main research interests include network security, artificial immune systems, cloud computing, and cloud storage. He has published nearly 300 papers in IEEE, ACM, Chinese Science, Science Bulletin, Natural Science Progress and other important journals and academic conferences.