Knowledge-Based Systems 190 (2020) 105528

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Detection of SQL injection based on artificial neural network☆

Peng Tang [a], Weidong Qiu [a,*], Zheng Huang [a,b], Huijuan Lian [a], Guozhen Liu [a]

[a] School of Cyber Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[b] Westone Cryptologic Research Center, Beijing, China

ARTICLE INFO

Article history:
Received 26 February 2019
Received in revised form 5 January 2020
Accepted 13 January 2020
Available online 16 January 2020

Keywords:
SQL injection
Neural network
MLP
LSTM

ABSTRACT

The SQL injection, a common web attack, has been a challenging network security issue which causes annually millions of dollars of financial loss worldwide as well as a large amount of users' privacy data leakage. This work presents a high accuracy SQL injection detection method based on neural network. We first acquire authentic user URL access log data from the Internet Service Provider(ISP), ensuring that our approach is real, effective and practical. We then conduct statistical research on normal data and SQL injection data. Based on the statistical results, we design eight types of features and train an MLP model. The accuracy of the model maintains over 99%. Meanwhile, we compare and evaluate the training effect of other machine learning algorithms(LSTM, for example), the results reveal that the accuracy of our method is superior to the relevant machine learning algorithms.

1. Introduction

Cyber-attacks cost the economy nearly $50 billion annually on average, and more than a fifth of which is caused by SQL injections. According to the survey, sampling 300,000 attacks worldwide in a given month, of which 24.6% were SQL injections [1,2]. In the 2014 Global Threat Intelligence Report released by NTT Corporation, it pointed out a cruel fact, the average after-care expenses of a small-scale SQL injection attack by a company usually exceeds $196,000 (over 1.2 million yuan). SQL injection attacks will not stop in a short period of time, and will continue to exist along with the development cycle of computer technology. It is an imminent and costly major threat. Therefore, deploying a web application attack defense strategy including SQL injection attacks is a necessary and primary security task, which is a key step to protect enterprise data as well as user privacy data.

SQL injection attacks are achieved by inserting SQL commands into a Web form, domain name, or page request, and finally tricking the server to execute malicious SQL commands, causing great harm to the website and users [3]. On the one hand, attackers can simply inject statements into the user input box to make the logical judgment constant, thus bypassing the access control of the server to unauthorized users, which not only causes the leakage of users' privacy data, but also may bring huge economic losses to users; On the other hand, attackers can inject incorrect SQL statements, resulting in grammatical or logical errors in the database, causing the database to crash and unable to provide data to the website normally.

Traditional SQL injection defense method uses blacklist filtering, which stores a blacklist and regular expressions to filter keywords or illegal strings. However, this method cannot filter out keywords and strings outside the blacklist, leaving web programs vulnerable to new SQL injection attacks.

This paper presents a simple and efficient method for SQL injection detection based on artificial neural network, it consists of three parts: data preprocessing, feature extraction and model training. Firstly, 8 types of relevant features are extracted by analyzing a large amount of SQL injection data, and then a great many actual data is used to train the neural network model. Finally, model training results are compared. In this paper, a variety of neural network models are used for training. MLP adopts the artificial feature extraction method to extract the corresponding features from the URL as the model input. LSTM is also used for comparison experiments. LSTM directly converts the URL into a vector as the model input. The experimental results show that the detection effect of MLP is slightly better than that of LSTM.

The rest of the paper is organized as follows: related work is covered in Section 2. Section 3 provides a brief overview of the proposed work. Experimental setup and result analysis are discussed in Section 4, while Section 5 concludes the paper with the future scope.

☆ No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.knosys.2020.105528.

* Corresponding author.
E-mail address: qiuwd@sjtu.edu.cn (W. Qiu).

https://doi.org/10.1016/j.knosys.2020.105528
0950-7051/

2. Related work

2.1. Traditional SQL injection detection method

Traditional SQL injection defenses use blacklists to filter out illegal characters, and much has been done in this area. Whitelist validation and blacklist validation are two different types of input validation approaches to defense SQL injection in the book named "SQL Injection Attacks and Defense" [4]. Similar to the blacklist method, the string-matching algorithm is used to calculate how well a string in a URL matches an SQL injected string to determine whether there is an SQL injection attack [5]. The penetration testing can be used to make up for the shortcomings of black and white list filtering defense mechanism, but it cannot fundamentally solve the defects [6]. Xiang et al. [7] proposed a static analysis framework named SAFELI for identifying SQL injection vulnerabilities on compile stage. SAFELI can use symbolic execution to inspect MSIL bytecode of an ASP.NET Web application statically. On the basis of traditional blacklist technology, Halfond and Orso [8] developed a tool, AMNESIA, that implemented a dynamic and static combination method, this technology automatically built a model of the legitimate queries that could be generated by the application in static part. In its dynamic part, it inspected the dynamically-generated queries and checked them against the statically-built model. Huang et al. [9] introduced a new web vulnerability scanner that used penetration testing and combinative evasion techniques to evade firewalls and filters and to detect injection. Masri and Sleiman [10] proposed an effective, light, and fully automated tool, SQL injection Prevention by Input Labeling (SQLPIL), utilized prepared statements to prevent SQLIAs at runtime. John [11] analyzed the advantages and disadvantages of the existing techniques against SQL injection and proposed a novel and effective solution to avoid attacks on login phase. Parvez et al. [12] analyzed the performance and detection capabilities of latest black-box web application security scanners against stored SQLI and stored XSS and developed the custom test-bed to challenge the scanners' capabilities to detect them.

2.2. Machine learning method

Many researches have applied machine learning methods to web attack detection, the most common web attacks include SQL injection, cross site scripting (XSS), denial of service, phishing, spamming, URL misinterpretation, etc. [13]. The essence of Web attack detection is a binary problem, a web site or a URL can only be malicious or normal. Machine learning techniques are used for the automated classification of Web pages [14]. Three major classifiers SVM, Random Forest and Naive Bayes are used for phishing website detection [15]. The Cuckoo Search SVM model is used to detect phishing emails. The CS-SVM extracts 23 features, which are used to construct the hybrid classifier [16]. A layered solution has been used to detect phishing web pages, named "CANTINA+", it is trained on the SVM model to detect phishing attacks [17].

There are also many researches on SQL injection attack detection using traditional machine learning, Joshi et al. [18] devised a classifier which consists of Naïve Bayes machine learning algorithm and Role Based Access Control mechanism for detection of SQL injection attacks, and the model is tested by three SQLIA attacks: comments, union and tautology. Kamtuo and Soomlek [19] proposed a SQL injection prevention framework based on decision tree and used 1100 vulnerability data set to train the machine learning model. Wu and Chan [20] proposed a method named k-centers (KC) to detect SQL injection attacks, it adapts to different types of attacks by adjusting the number and location of clusters in KC, which is a clustering algorithm of mixed type data based on k-means [21], as well as a traditional machine learning algorithm.

2.3. Neural network

In recent years, deep neural network, also known as deep learning [22], has gradually emerged. Deep learning is an algorithm in machine learning. Artificial Neural Network can find complex structures in big data set by simulating the interconnecting structure of neurons in human brain. The main difference between deep learning and traditional machine learning is that its performance increases as the data size increases. Deep learning algorithms do not perform well when data is scarce, this is because it requires large amounts of data to train. The research of web security detection using deep learning method also has some achievements. Wang et al. [23] presented a deep learning framework for detection of malicious JavaScript code, the framework is composed of a sparse random projection, deep learning model, and logistic regression. The model used 27 000 labeled samples, and its accuracy rate was more than 95%, while the false alarm rate was less than 4.2%. Sirinam et al. [24] used a fingerprint attack on the CNN defense website, the results show that the accuracy of CNN's fingerprint attack detection on the website is more than 98%. Yuan et al. [25] provided a deep learning enabled subspace spectral ensemble clustering approach, called DEP-SSEC, for web attack detection. Selvaganapathy et al. [26] used deep belief network to extract features of URLs and deep neural network to classify normal and malicious URLs.

Based on the above research, this paper proposes an artificial neural network SQL injection attack detection model based on deep learning, which uses actual pipeline data provided by ISP to train neural networks (including LSTM and MLP) models. The model can effectively and comprehensively detect SQL injection attacks. Compared with the traditional black and white list rule filtering method, this method can learn the general rules of SQL injection through neural network, and can effectively solve the problem of incomplete blacklist filtering. Based on the paper [27], we extracted more features for SQL injection detection, which greatly improved the accuracy of SQL injection detection. Meanwhile, more experimental comparisons were made.

3. SQL injection prevention using neural network

In this section, we will introduce the algorithm design of SQL injection detection using neural network. Due to the limited information in the data set, we will only extract and classify the URL features. Extracting more characteristic information besides URL (such as page information, cookie information) will be the future research direction.

3.1. System framework

This paper uses big data analysis method to analyze normal URL and SQL injection URL, so as to discover the URL feature of SQL injection, based on which, neural network is used for binary training, the trained model is deployed to the big data platform of the ISP to carry out SQL injection detection on the HTTP traffic data of the user in real time. Thus, a complete SQL injection detection system based on neural network is implemented. The overall framework of the system is shown in Fig. 1.

The metadata of the entire system comes from the DPI data provided by the ISP. We use these data for SQL injection analysis, including data processing, model training, and model evaluation.

The data processing layer pre-processes the metadata, including feature extraction, standardization, data division, and so on. Model training is based on computing platforms, such as GPUs. We use MLP and LSTM to perform SQL injection detection training on the extracted features. The model is evaluated using standards such as accuracy, precision, recall, and FPR. We deploy the trained model in the ISP system, so that we can find abnormal behaviors in the network in time.

Table 1
Features.

Features	Value	Calculation method	Descriptions
LP	Numeric	len(payload)	Length of payload in URL
NK	Numeric	$\sum_{k \in keywords}$ payload(k)	Number of keywords in payload
KWS	Numeric	$\sum_{k \in keywords}$ Weight[k] × payload[k]	Sum of keywords weight in payload
NSPA	Numeric	payload [space] + payload["%20"]	Number of Spaces in payload
RSPA	Float	NSPA/LP	Ratio of spaces
NSPE	Numeric	$\sum_{s \in special\ characters}$ payload(s)	Number of special characters
RSPE	Float	NSPE/LP	Ratio of special character
ROC	Float	1 − PSNA − PSNE	Ratio of other characters

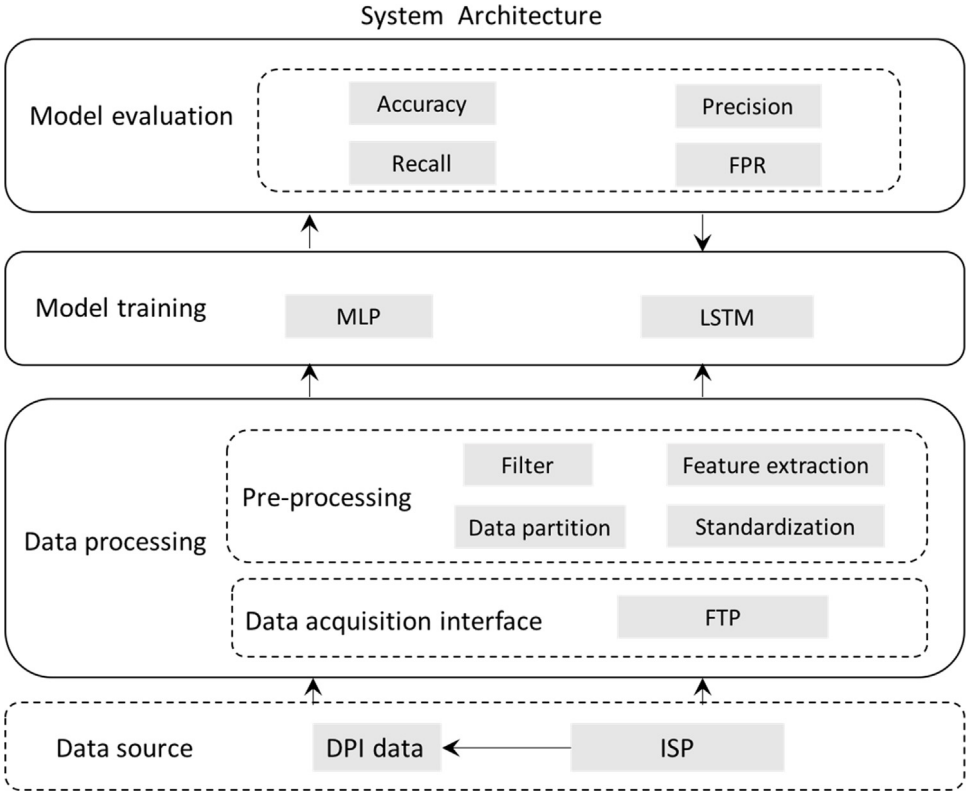

Fig. 1. The overall framework of system.

3.2. Features

SQL injection attacks have different features from normal URLs. Not only do they include sql-related keywords, but the form of the parameters is more complex than normal parameters, this is due to the attacker's careful construction of parameters, so that parameters can be successfully executed to achieve the malicious purpose after arriving at the server.

In this article, we extracted eight character features and character statistical features of URLs as input of the neural network, we will discuss these 8 features in detail. (See Table 1.)

(1) Length of Payload

Payload is part of the URL and is usually in the form of key–value pairs of parameters, URLs containing SQL injections are often loaded with malicious SQL scripts, therefore, the paper mainly extracts and analyzes payload in URLs.

Payload that contains SQL injection typically contains SQL statements, so payload is generally longer than the payload of a normal URL, in this paper, payload character length is recorded as LP as a feature.

$$LP = \text{len}(payload) \tag{1}$$

(2) Number of Keywords

SQL statement keywords is the key to SQL injection statements. Injection statements without keywords can hardly affect the normal operation of the database even if they are passed to the back end. Therefore, this article primarily analyzes whether the URL contains SQL statement keywords.

Common keywords for SQL injection are shown in Table 2.

The number of keywords is marked as NK,

$$NK = \sum_{i \in keywords} payload(k) \tag{2}$$

Payload (k) represents the number of the keyword k in the payload.

(3) Sum of Keywords Weight

Some of the above keywords are not unique, such as "update" and "count", which are often used in statements outside the

Table 2
Keywords for SQL injection.

Keyword category	Keyword
Data query	select, union, count, group by, order by
Data modification	insert, delete, update, drop table, truncate table
Connection symbol	and, or, where, from, into
System operation	exec, xp_cmdshell, master, net
File operation	load_file, outfile, dumpfile

Table 3
Weights of keywords.

Weight	Keyword
5	union, truncate, xp_cmdshell, load_file, outfile, dumpfile, exec
3	select, update, insert, delete, count, where, group, order, drop, table, master, net
1	and, or, by, from, into

database and may appear in the form submission parameters. Typically, it cannot determine whether the URL statement is suspicious when there is only one or two keywords. Therefore, we set different weights to the above keywords.

In order to facilitate analysis, the combined keywords are disassembled, so that the weight of the disassembled keywords is reduced, yet the weight sum is still large if it occurs at the same time. The allocation of weights is based on experience of assigning relatively low weights to frequently used words and relatively high weights to infrequently used words, with a maximum weight of 5 and a minimum of 1. In this article, the SQL keywords in the parameter values are weighed sum, and the results are taken as the first feature.

Weights of keywords are shown in Table 3.

Total weight of keywords is marked as KWS,

$$\mathrm{KWS} = \sum_{k \in \mathrm{keywords}} \mathrm{Weight}[k] \times \mathrm{payload}[k] \tag{3}$$

Among which, Weight [k] represents the Weight of the keyword k and payload[k] represents the number of the keyword k in the payload.

(4) Number of Spaces

In most cases, the URL parameter is a number or a simple string with zero or fewer spaces. Although there are also a few cases where long information is passed using URL, the inclusion of SQL statements in the URL makes the number of spaces significantly more than the normal URL.

Under normal circumstances, if the URL parameter contains non-alphanumeric characters, these special characters will be converted into % prefix characters, and the data will be transmitted to the server for decoding. For spaces, it will be converted to "%20", and then we analysis the number of the "%20" strings and spaces,marked as NSPA,

$$\mathrm{NSPA} = \mathrm{payload}\,[\mathrm{space}] + \mathrm{payload}[\text{“}\%20\text{”}] \tag{4}$$

(5) Ratio of Space

Calculate the ratio of space length as an input feature, marked as RSPA,

$$\mathrm{RSPA} = \mathrm{NSPA}/\mathrm{LP} \tag{5}$$

(6) Number of Special Characters

On the other hand, the attacker constructs SQL statements with annotations, which change the structure of the original SQL when the parameters are passed to the server, making the query conditions smaller and thus gaining greater authority. Commonly used comments in SQL statements include "- -", "#", "/**/", etc. These symbols will increase the possibility of URL injection for SQL to some extent. Therefore, we analyze the percentage of special characters in the data as a percentage of the length of the URL parameter values.

Table 4
Special characters.

Characters	Instructions
+, −, ∗, /	For operation
=, ! =, ^=, <>, >= ,<=	For assignment and conditional judgment
- -, #, /∗, ∗/	For notes
‘’, “”, @, \, ()	Other

Special characters are shown in Table 4.

Number of special characters recorded as NSPE,

$$\mathrm{NSPE} = \sum_{s \in \text{special characters}} \mathrm{payload}(s) \tag{6}$$

Wherein, payload(s) represents the number of special characters s in the payload.

(7) Ratio of Special Characters

The ratio of the length of special characters is calculated as an input feature, marked as RSPE,

$$\mathrm{RSPE} = \mathrm{NSPE}/\mathrm{LP} \tag{7}$$

(8) Ratio of Other Characters

Calculating the proportion of the payload of characters other than spaces and special characters as an input feature, marked as ROC,

$$\mathrm{ROC} = 1 - \mathrm{PSNA} - \mathrm{PSNE} \tag{8}$$

3.3. Models

SQL injection detection is a binary problem, we classify the URL characteristics entered as malicious and non-malicious and use several neural network models to solve this problem including MLP and LSTM.

3.3.1. MLP

Multi-Layer Perceptron (MLP) is a feedforward artificial neural network with multiple processing layers, includes an input layer, an output layer, and at least one hidden layer, among which, both the hidden layer and the output layer have processing ability. Multilayer perceptron with a single hidden layer can be regarded as a special logistic regression classifier, it first applies nonlinear transform to the input data, and then takes the result as the input of logistic regression. The nonlinear transformation causes the input samples to be mapped to a space where they are linearly separable [28].

In MLP, the input layer does not participate in information processing, it consists of multiple input nodes, which obtain external input without any calculation, and only pass information to hidden nodes.

The hidden layer consists of multiple hidden nodes that have no external connection. The output of the hidden node depends on the output of the input layer and the weight attached to the connection, and passes the result to the next hidden or output layer. A MLP network may contain one or more hidden layers.

The output layer consists of multiple output nodes that obtain data from the hidden layer and perform computations similar to the hidden layer. The end result is the output of the MLP, delivered to the outside world.

As shown in Fig. 2, this network has three layers, among which L_1 is the input layer, L_2 is the hidden layer, and L_3 is the output layer. There are full connections between L_1 and L_2, and between L_2 and L_3 nodes.

In this structure, $w_{ij}^{(l)}$ represents the weight of unit j in layer l and unit j in layer l+1, $b_i^{(l)}$ represents the bias of unit i+1 in layer

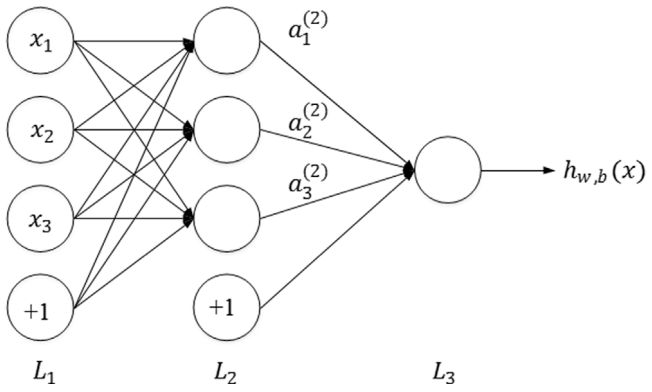

Fig. 2. The structure of the MLP.

l, $a_i^{(l)}$ represents the activity of unit i in layer l+1, $h_{w,b}$ represents the final output. The calculation process of the structure is as follows:

$$a_1^{(2)} = f(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + b_1^{(1)}) \quad (9)$$

$$a_2^{(2)} = f(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3 + b_2^{(1)}) \quad (10)$$

$$a_3^{(2)} = f(w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3 + b_3^{(1)}) \quad (11)$$

$$h_{w,b}(x) = f(w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)} + b_1^{(2)}) \quad (12)$$

Among which, f is the activation function. The commonly used activation functions are ReLU, sigmoid, tanh and softplus. The function curve is shown in Fig. 3.

$$relu(x) = max(0, x) \quad (13)$$

$$sigmoid\,(x) = \frac{1}{1+e^{-x}} \quad (14)$$

$$\tanh\,(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad (15)$$

$$softplus\,(x) = \ln(1+e^{x}) \quad (16)$$

This paper chooses a more general ReLU function as the activation function, because the ReLU function does not activate all neurons, making the neural network efficient and easy to calculate. At the same time, different activation functions were used in this paper for comparative experiments. The experimental results are shown in Fig. 4. The experimental results show that in the application scenario of this paper, the detection results of different activation functions have little difference, and the ReLU function is slightly better than other activation functions.

The essence of multilayer perceptron is complex function fitting, and the classification of nonlinear data can be realized by multilayer linear model. Take the two-layer perceptron as an example, from the input layer to the hidden layer, the hidden layer realizes the spatial transformation of the original data, making it linearly separable, and then classifies it in the output layer.

3.3.2. LSTM

The MLP-based SQL injection detection proposed above is to distinguish malicious and normal URLs by analyzing the statistical features of keywords, spaces and special characters in URL parameters. It reflects the character statistics features but not the syntax features of malicious SQL statements. In fact, malicious SQL statements must be organized according to the rules of SQL so that malicious parameters can be passed to the server to work and achieve malicious purposes. However, the rules of SQL syntax are reflected in the order in which keywords, Spaces, and special characters are used. Although SQL syntax can be used flexibly, it always shows regularity. Locally, nevertheless, each SQL keyword is composed of smaller characters that are sequentially arranged to form keywords that cannot be formed by other permutations or other permutations of characters. Thus, a string can be thought of as a sequence of characters. A sequence of characters in a malicious SQL statement presents a sequence features different from that of a normal statement. SQL injection detection can be performed by learning the sequence of malicious samples.

RNN is a deep learning model suitable for solving sequential problems. Considering the length of SQL injection statements, LSTM [29] is also adopted in this paper as a special RNN [30] to

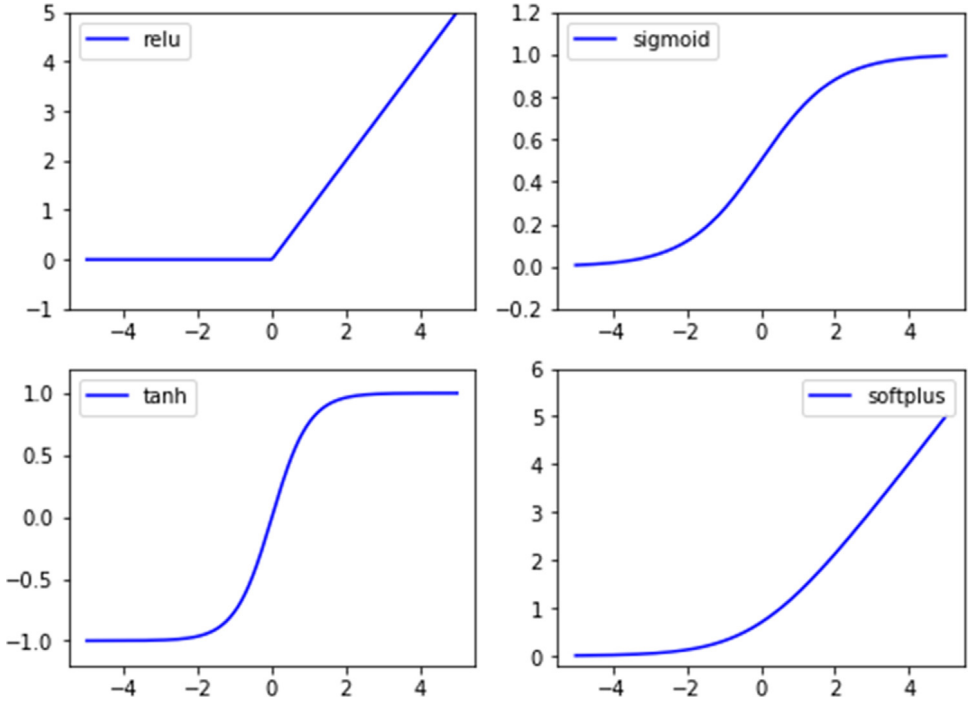

Fig. 3. Activation function.

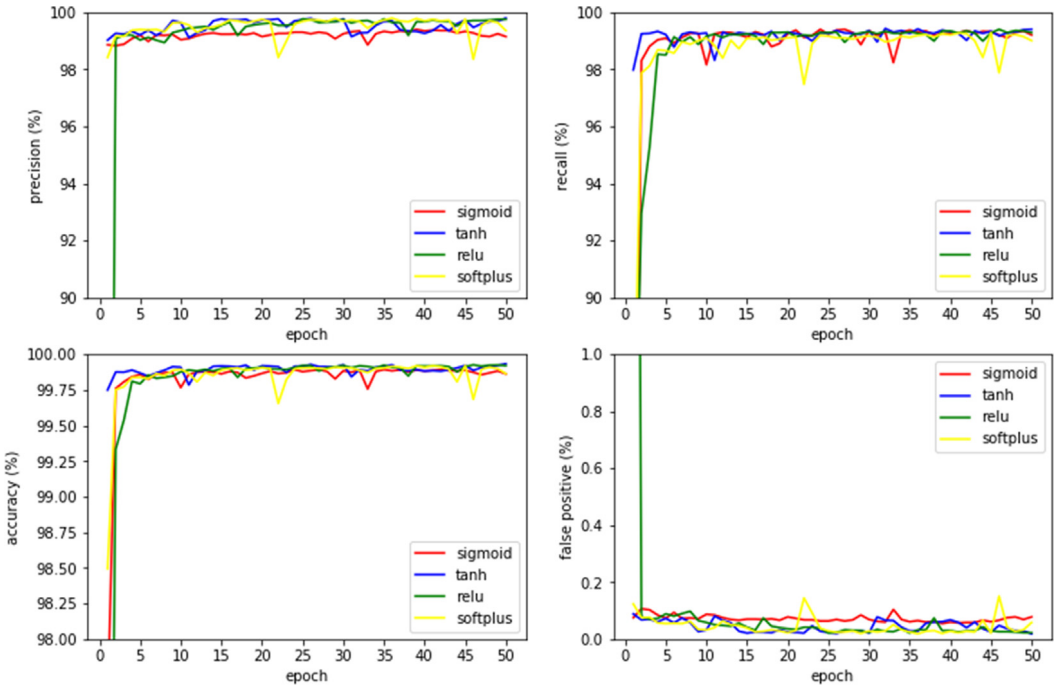

Fig. 4. Experiment results of different activation function.

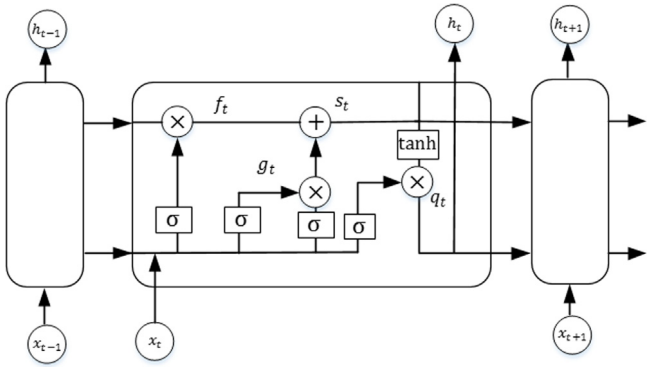

Fig. 5. The structure of the LSTM.

realize long-term memory of characters. The LSTM structure is shown in Fig. 5.

The figure is an LSTM, where the state unit is a linear self-cyclic structure, and its cyclic weight is controlled by the forgetting gate. The forgetting gate receives the hidden layer output of the previous time and the external input of the current time and optionally forgets, and then returns the self-cyclic weight. The calculation is as follows:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_i^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right) \tag{17}$$

Among which, $x_i^{(t)}$ is the input vector at time t, $h_j^{(t-1)}$ is the vector of hidden layer at time $t-1$, $U_{i,j}^f$ is the weight of input, $W_{i,j}^f$ is the forgotten weight, b_i^f is the bias and σ is the sigmoid function.

The input gate is updated in a similar way to the forgetting gate, and the calculation is as follows:

$$g_i^{(t)} = \sigma(b_i^g + \sum_j U_{i,j}^g x_i^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}) \tag{18}$$

The update of the state unit is related to the input gate and forgetting gate, so the state at time t is calculated as follows:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(b_i + \sum_j U_{i,j} x_i^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}) \tag{19}$$

The output gate is calculated as follows:

$$q_i^{(t)} = \sigma(b_i^o + \sum_j U_{i,j}^o x_i^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}) \tag{20}$$

Finally, the output $h_i^{(t)}$ of LSTM at time t is controlled by the state unit and the output gate, and is calculated as follows:

$$h_i^{(t)} = \tanh(s_i^{(t)}) q_i^{(t)} \tag{21}$$

4. Experimental results and analysis

In this section, we evaluate the effectiveness of different neural network for SQL injection detection. We used the actual data provided by the ISP, so that the model we trained could be deployed in the actual environment.

4.1. Experimental environment

The software environment of the algorithm is Ubuntu 16.04 operating system and Pytorch deep learning framework, and the programming language is Python.

Hardware platform is a dell PowerEdge R730 server (CPU model Xeon e5-2640 v3, 64 GB memory, 1.2t hard disk, equipped with NVIDIA Tesla M40 GPU).

4.2. Model implementation

Pytorch is used to construct MLP and LSTM neural network.

4.2.1. MLP

For the MLP model, we use 8 features as input, that is, the number of nodes in the input layer of MLP is 8. For the features

Table 5
Hyperparameter of MLP.

Hyper Parameter	Description	Setting
input_size	Number of input layer nodes	8
hidden_size	Number of hidden layer nodes	100
output_size	Number of output layer nodes	2
num_epoch	Number of epoch	50
batch_size	Batch size	1000
learning_rate	Learning rate	0.01

with numerical type, after normalization, it is used as input. Min–max normalization is used for normalization. The formula is as follows:

$$x = \frac{x - min}{max - min} \tag{22}$$

Where max and min represent the maximum and minimum values of sample data respectively.

The hyperparameter settings for the MLP model are shown in Table 5. We extracted a total of 8 features as input, so our input_size was set to 8, the hidden layer adopted 100 points, and the output layer was 2 nodes. Batch size and epoch of the neural network were set as 1000 and 50 respectively, and the learning rate was set as 0.01.

In addition, we select ReLU as the activation function, use CrossEntropyLoss as our loss function, and use Adam as the optimization algorithm.

4.2.2. LSTM

For the LSTM model, we first convert the character sequence into a numeric matrix. This paper uses ASCII code to map characters. ASCII code, the American standard code for information interchange, is a coding standard. In ASCII code tables, letters, spaces, and special characters can all be represented as corresponding numbers. For example, the SQL statement "select * from tb_user" can be converted to an array of length 21 based on the ASCII code:

[115, 101, 108, 101, 99, 116, 32, 42, 32, 102, 114, 111, 109, 32, 116, 98, 45, 117, 115, 101, 114]

Second, we unify the sequence length. In general, the length of SQL injection statements is almost always different, and the sample dimensions received by the deep learning model must be the same. According to the statistics of the length of malicious data, 95% of the SQL injected strings are less than 100 in length, so 100 is selected as the uniform sequence length. A sequence more than 100 gets truncated to 100, while a sequence less than 100 gets padding zero. Eventually each of these will turn into a 1×100 matrix:

$[115, 101, 108, 101, \ldots, 0, 0, 0]$

Finally, we transpose it to a 100×1 matrix. According to the principle of LSTM, the expansion of the model is equivalent to reading a 1×1 vector at each time point, with a total of 100 time points:

$[115, 101, 108, 101, \ldots, 0, 0, 0]^T$

The network parameters of LSTM are shown in Table 6. Hyper parameters of the LSTM model are similar to MLP's with an addition of time_ step. The time_ step is a time step, which is a unique super parameter of LSTM. It represents the length of time window, that is, the model will remember the data within the time range of t-time_ step $\sim$ t at time t. For this experiment, time_ step indicates that the processed character length is 100.

Table 6
Hyperparameter of LSTM.

Hyper Parameter	Description	Setting
input_size	Number of input layer nodes	1
hidden_size	Number of hidden layer nodes	100
output_size	Number of output layer nodes	2
time_step	Time step	100
num_epoch	Number of epoch	50
batch_size	Batch size	1000
learning_rate	Learning rate	0.01

4.3. Experimental data

The required malicious data is from the parameter payload of SQL injection provided by the open source website Github, and the normal data is from the ISP. The malicious data of training sets and test sets are distributed in a ratio of 7:3; The normal data and malicious data in the test sets are 1:1. We have 7095 pieces of malicious data and 76960 pieces of normal data in the training set, and 3040 pieces of both normal data and malicious data in the test set.

The malicious data samples are payloads of the parameters after URL parsing, which are as follows:

```
323%27%20AND%20%28%20SELECT%202937%20FROM%28%20SELECT%20COUNT%28%2A%29%2C%20CONCAT%280x3a6d70663a%2C%28%20SELECT%20MID%28%28%20IFNULL%28%20CAST%28%20database%28%29%20AS%20CHAR%20%29%2C0x20%29%29%2C1%2C50%29%29%2C0x3a736e623a%2CFLOOR%28RAND%280%29%2A2%29%29x%20FROM%20INFORMATION_SCHEMA.CHARACTER_SETS%20GROUP%20BY%20x%29a%29%20AND%20%27rmHN%27=%27rmHN
```

After the URL is decoded, you can see the obvious SQL injection statement.

```
323' AND ( SELECT 2937 FROM( SELECT COUNT(*), CONCAT(0x3a6d70663a,( SELECT MID(( IFNULL( CAST( database() AS CHAR ),0x20)),1,50)),0x3a736e623a,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a) AND 'rmHN'='rmHN
```

The full URL of the normal data sample is shown below:

```
http://nl.rcd.iqiyi.com/apis/mbd/upload.action?agent_type=20&version=8.11.0&ua=iPhone8,2&network=1&os=11.0.3&com=1&wsc_istr=3CE76AD3-3291-4625-9DC6-5163437FE4F6&wsc_lgt=118.83249&wsc_ltt=31.91966&wsc_tt=02&wsc_ost=13&wsc_osl=zh-Hans-CN&wsc_st=iQiYiPhoneVideo8.11.0
```

The analysis of URL text is mainly to analyze the URL parameter strings. According to URL naming rules, a complete URL includes protocol, domain name, port, directory, file, parameter. Where the parameter part is represented in the form of a key–value pair. The key and value are separated by "=" and different key pairs are separated by "&". Take the following URL as an example:

http://hdns.ksyun.com/d?dn=jsmov2.a.yximgs.com&ttl=1

The parameter part is "dn=jsmov2.a.yximgs.com&ttl=1", indicating that the URL contains two parameters "dn" and "ttl", and the parameter values are "jsmov2.a.yximgs.com" and "1" respectively. The analysis of URL text in this article is the analysis of each parameter value.

Table 7
TP, FP, FN & TN.

	Predict value	
Actual value	Positive	Negative
Positive	True Positive (TP)	False Negative (FN)
Negative	False Positive (FP)	True Negative (TN)

Table 8
Training results.

Model	acc	precision	recall	FPR	time(s)
MLP-1 hidden layer	99.86%	99.55%	98.80%	0.04%	94.4
MLP-2 hidden layer	99.88%	99.38%	99.17%	0.06%	111.2
MLP-3 hidden layer	99.92%	99.76%	99.30%	0.02%	126.8
LSTM	98.69%	99.85%	95.69%	0.06%	10692.1

4.4. Evaluation

The results of the model for the test set data and the real labels of the data contain four combinations: TP, FP, FN, TN, as shown in Table 7.

We evaluated the effect of neural network model from the following four aspects on the training set and test set:

Accuracy, represents the proportion of the correct sample number in the total number of samples:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{23}$$

Precision, indicates how many of the samples predicted to be positive are truly positive:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{24}$$

Recall, indicates how many positive samples in the sample were predicted correctly:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{25}$$

FPR, Predict the ratio of negative samples to positive samples:

$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}} \tag{26}$$

4.4.1. Training results

In the training set, accuracy, precision, recall and FPR of the model change with the epoch as shown in Figs. 6 and 7.

The final training results of the model are shown in Table 8. In this article, TP + FN is the total number of malicious samples, and FP + TN is the total number of normal samples.

As shown in Fig. 6, we use MLP neural networks with different depths (1, 2, and 3 layers of hidden layers) for training. The training results show that with the increase of epoch, the training effect is getting better and better, and gradually stabilizes. With the increase of neural network hiding layer, the training effect also improves slightly. The accuracy of MLP with three layers of hiding layer is as high as 99.92%, but relative to MLP with one layer of hidden layer, it is only increased by 0.06%. As shown in Fig. 6, we simultaneously use LSTM neural network for comparative experiments. The accuracy and recall rate of LSTM is only 98.69% and 95.69%, lower than the training effect of MLP. In terms of training time, the model adopts NVIDIA Tesla M40 for accelerated calculation. With the increase of hidden layer, the training time of the model also increases. Under the condition of training the same data set, the training time of MLP model is around 100 s, while the training time of LSTM is much longer than that of MLP, up to 10^4s, which is about 100 times that of MLP.

Table 9
Test results.

Model	acc	precision	recall	FPR	time(ms)
MLP (1 hidden layer)	99.51%	99.97%	99.05%	0.03%	66.3
MLP (2 hidden layer)	99.36%	100.00%	98.72%	0.00%	71.5
MLP (3 hidden layer)	99.67%	100.00%	99.41%	0.00%	74.0
LSTM	97.68%	99.86%	95.49%	0.13%	37149.4

Table 10
Comparison based on accuracy.

Algorithms	Accuracy (%)
Naïve–Bayes Algorithm [18]	93.3%
SVM [31]	96.47%
SVM + SMO [32]	95.67%
SVM + PSO [33]	91.57%
Our approach	99.67%

In our original conference paper, the accuracy of MLP was 99.55%, the accuracy was 97.44%, the recall was 97.21%, and the FPR was 0.24%. The accuracy, precision, recall, and FPR of LSTM were: 95.00%, 99.25%, 90.69%, 0.69%. Based on this improvement and comparative experiments, the accuracy, precision, and recall of MLP increased by 0.37%, 2.38%, 2.15%, and FPR decreased by 91.67%. The accuracy, precision, and recall of LSTM increased by 3.88%, 0.60%, and 5.51%, while FPR decreased by 91.30%.

4.4.2. Test results

On the test set, a total of 6080 samples were tested, of which 3040 were malicious samples and 3040 were normal samples. The MLP model with 1 hidden layer correctly identified 3011 normal samples and 3039 malicious samples, while only 1 malicious sample was wrongly identified as normal samples. And the MLP model with 3 hidden layers correctly identified 3022 normal samples and 3040 malicious samples, while no malicious samples were wrongly identified as normal samples. The LSTM model correctly identified 2903 normal samples and 3036 malicious samples, and a total of 4 malicious samples were wrongly identified as normal samples, as shown in Table 9. The test results show that the MLP detection effect using the 3-layer hidden layer is better than the 1-layer MLP, the 2-layer MLP and the LSTM. Simultaneously detecting the same number of URLs, the MLP of the 3 layers of hidden layers takes 11% more time than the 1 layer of MLPs, and the detection time of LSTM is far more than MLP. In terms of running time, with the increase of the number of hidden layers, the time spent by the model to detect a piece of data is also longer. Among them, it takes an average of 66.3 ms for MLP of 1-layer to detect a piece of data, 74 ms for MLP of 3-layer to detect a piece of data, and 37149.4 ms for LSTM to detect a piece of data, which shows that the detection time of LSTM is much longer than that of MLP.

In terms of accuracy, we use the method proposed by us to compare with some advanced machine learning methods. Table 10 shows the comparison with Naïve-Bayes Algorithm, SVM, SVM+SMO, SVM+PSO proposed by some scholars. By comparison, the accuracy of our proposed method is much higher than other methods.

4.4.3. Validation results

We use public data sets to verify our experimental results. The public data set is from GitHub [34] and contains a total of 9745 samples, including 820 SQL injection samples and 8925 normal samples. All normal samples were collected from the Alexa website and have been verified. The malicious samples include In-band SQL injection, Error-based SQL injection, Time-based SQL injection and other types of SQL injection data. The verification results are shown in Table 11.

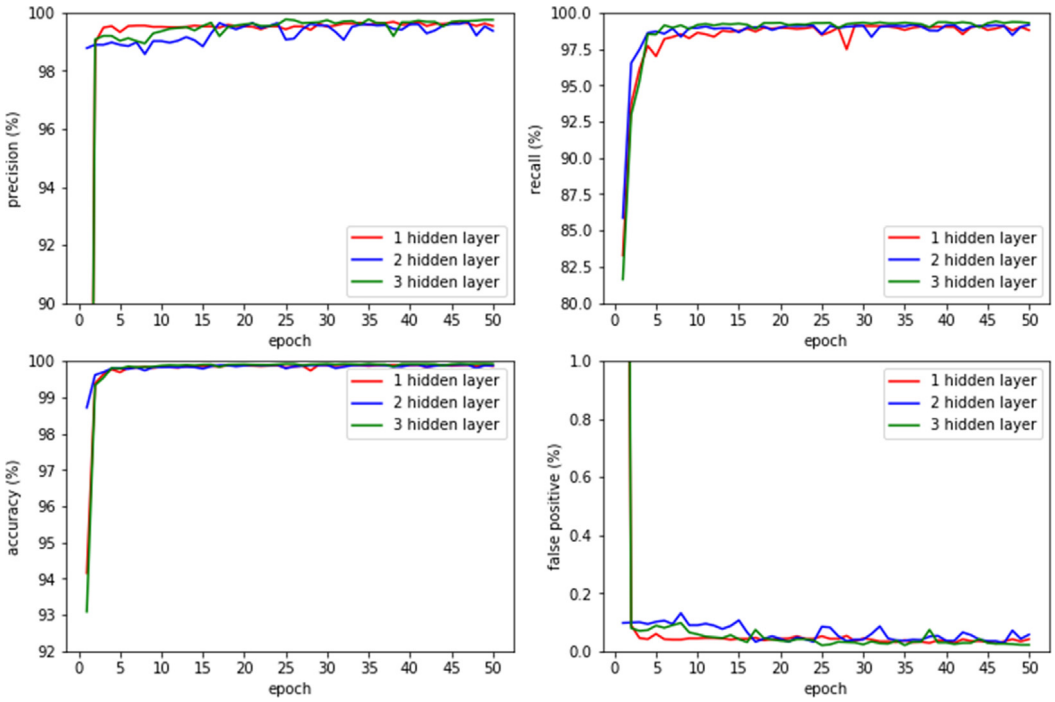

Fig. 6. MLP training results.

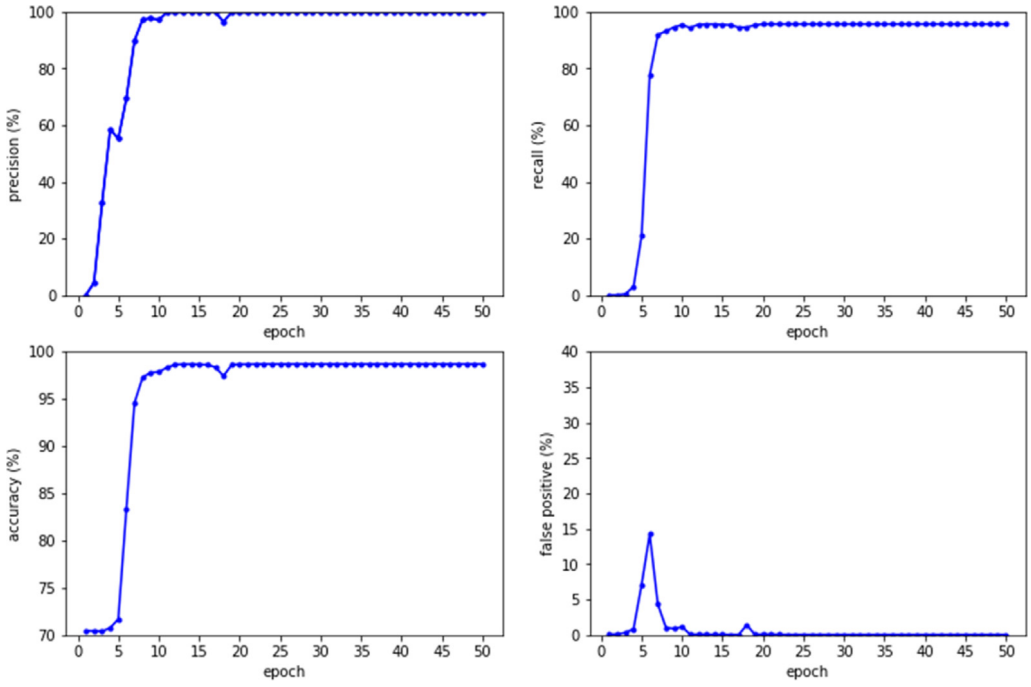

Fig. 7. LSTM training results.

Table 11
Validation results.

Model	acc	precision	recall	FPR	Time (ms)
MLP (1 hidden layer)	99.67%	96.24%	100%	0.36%	84.1
MLP (2 hidden layer)	99.55%	95.75%	99.02%	0.40%	103.2
MLP (3 hidden layer)	99.75%	97.27%	99.88%	0.26%	113.4
LSTM	99.17%	91.10%	99.88%	0.90%	45237.1

In the verification set, the 1-hidden layer MLP correctly identified all the malicious samples, whereas 32 normal samples were misidentified. As for the 3-hidden layer MLP, although one malicious sample was misidentified, only 23 normal samples were incorrectly identified. Since the total number of malicious samples in the verification set is small, according to formula 23–26, the recall rate of model is thus higher. To sum up, as can be seen from the test results of the verification set, the method proposed in this paper also performs well on unknown data, and the accuracy of the model remains above 99.5%.

5. Conclusion

This paper proposes a SQL injection detection framework based on neural network, which uses a variety of neural networks to detect SQL injection respectively. The SQL injection detection model based on MLP firstly extracts the corresponding URL features as the input of neural network, and then carries out MLP network training, saving the model with the best training effect as the final model. The SQL injection detection model based on LSTM converts URL into vector and uses vector as the input of LSTM for model training. Experimental results show that the feature extraction method proposed in this paper is superior to other neural network models.

On account of the MLP model receives input with manually extracted features, including statistics for keywords, spaces, and special characters. These features have largely separated malicious SQL statements from normal statements, and MLP has achieved good detection results by learning these features. However, in the detection of LSTM model, the recognition ability of specific keywords and characters is poor, and it pays attention to the recognition of the relationship between keywords and special characters, as well as the internal characters of keywords. In addition, ASCII code is adopted for the processing of strings in the experiment. In ASCII code, special characters and letters are similar, and the distinction of numerical size is not obvious, which also causes the blurriness of the boundary between letters and characters in the detection process of LSTM.

While there is a performance gap between LSTM and MLP, LSTM has unique advantages. LSTM takes advantage of the character order in the string and actively learns the association of characters before and after according to its internal structure, so as to distinguish SQL statements from normal statements. This feature takes advantage of the LSTM in the natural language world, where it can even determine the emotional tone of a statement based on its context.

LSTM still has great potential in threat intelligence detection. For other types of Web attacks, whether XSS (cross-site scripting attacks) or phishing sites, you can take advantage of the LSTM model as long as these attacks have unique characteristics in the URL text or HTTP request text. This means that LSTM does not need to extract different features from different types of threats, as MLP does. It can learn the characteristics of different threats with sufficient samples to train, which has a strong scalability.

Acknowledgments

This work was supported by the National Key Research and Development Program of China under Grand 2017YFB0802704 and National Natural Science Foundation of China under Grand 61972249.

References

[1] D. Morgan, Web application security – SQL injection attacks, Netw. Secur. 2006 (4) (2006) 4–5.
[2] Imperva's Web Application Attack Report.
[3] J. Clarke, SQL Injection Attacks and Defense, Elsevier Ltd Oxford, 2009.
[4] J. Clarke, Platform-level defenses, in: J. Clarke (Ed.), SQL Injection Attacks and Defense, Syngress, Boston, 2009, pp. 377–413, Chapter 9.
[5] G. Buja, K.B.A. Jalil, F.B.H.M. Ali, T.F.A. Rahman, Detection model for SQL injection attack: An approach for preventing a web application from the SQL injection attack, in: 2014 IEEE Symposium on Computer Applications and Industrial Electronics, ISCAIE, 2014, pp. 60–64.
[6] W. Tian, J. Yang, J. Xu, G. Si, Attack model based penetration test for SQL injection vulnerability, in: 2012 IEEE 36th Annual Computer Software and Applications Conference Workshops, 2012, pp. 589–594.
[7] F. Xiang, L. Xin, B. Peltsverger, S. Chen, Q. Kai, L.X. Tao, A static analysis framework for detecting SQL injection vulnerabilities, in: International Computer Software & Applications Conference, 2007.
[8] W.G.J. Halfond, A. Orso, AMNESIA: analysis and monitoring for neutralizing SQL-injection attacks, in: IEEE/ACM International Conference on Automated Software Engineering, 2005.
[9] H.C. Huang, Z.K. Zhang, H.W. Cheng, S.W. Shieh, Web application security: Threats, countermeasures, and pitfalls, Computer 50 (6) (2017) 81–85.
[10] W. Masri, S. Sleiman, SQLPIL: SQL injection prevention by input labeling, Secur. Commun. Netw. 8 (15) (2015) 2545–2560.
[11] A. John, SQL Injection Prevention by Adaptive Algorithm.
[12] M. Parvez, P. Zavarsky, N. Khoury, Analysis of effectiveness of black-box web application scanners in detection of stored SQL injection and stored XSS vulnerabilities, in: Internet Technology & Secured Transactions, 2016.
[13] G. Nivedhitha, Survey Paper in Detecting and Preventing the Web Application Vulnerability.
[14] M.I. Devi, D.R. Rajaram, K. Selvakuberan, Machine learning techniques for automated web page classification using URL features, in: International Conference on Conference on Computational Intelligence & Multimedia Applications, 2007.
[15] V. Preethi, G. Velmayil, Automated Phishing Website Detection Using URL Features and Machine Learning Technique.
[16] W. Niu, X. Zhang, G. Yang, Z. Ma, Z. Zhuo, [IEEE 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications, ISPA/IUCC - Guangzhou, China (2017.12.12-2017.12.15)] 2017 IEEE Internation, 2017, pp. 1054–1059.
[17] G. Xiang, J.I. Hong, C.P. Rosé, L.F. Cranor, CANTINA+: A feature-rich machine learning framework for detecting phishing web sites, ACM Trans. Inf. Syst. Secur. 14 (2) (2011) 21.
[18] A. Joshi, V. Geetha, SQL Injection detection using machine learning, in: International Conference on Control, 2014.
[19] K. Kamtuo, C. Soomlek, Machine Learning for SQL injection prevention on server-side scripting, in: Computer Science & Engineering Conference, 2017.
[20] X.R. Wu, P.P.K. Chan, SQL injection attacks detection in adversarial environments by k-centers, in: International Conference on Machine Learning & Cybernetics, 2012.
[21] W.D. Zhao, W.H. Dai, C.B. Tang, K-centers algorithm for clustering mixed type data, in: Advances in Knowledge Discovery & Data Mining, Pacific-Asia Conference, Pakdd, Nanjing, China, May, 2007.
[22] Y. Lecun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436.
[23] Y. Wang, W.D. Cai, P.C. Wei, A deep learning approach for detecting malicious JavaScript code, Secur. Commun. Netw. 9 (11) (2016) 1520–1534.
[24] P. Sirinam, M. Imani, M. Juarez, M. Wright, Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning, 2018.
[25] G. Yuan, L. Bo, Y. Yao, S. Zhang, A deep learning enabled subspace spectral ensemble clustering approach for web anomaly detection, in: International Joint Conference on Neural Networks, 2017.
[26] S. Selvaganapathy, M. Nivaashini, H. Natarajan, Deep belief network based detection and categorization of malicious URLs, Inf. Secur. J.: Global Perspect. 27 (3) (2018) 145–161.
[27] P. Tang, W. Qiu, Z. Huang, H. Lian, G. Liu, SQL Injection Behavior Mining Based Deep Learning, 2018.
[28] I.D. Longstaff, J.F. Cross, A pattern recognition approach to understanding the multi-layer perceptron, Pattern Recognit. Lett. 5 (5) (1987) 315–319.
[29] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[30] F.J. Pineda, Generalization of back-propagation to recurrent neural networks, Phys. Rev. Lett. 59 (19) (1987) 2229–2232.
[31] R. Rawat, S.K. Shrivastav, SQL injection attack detection using SVM, Int. J. Comput. Appl. 42 (13) (2012) 1–4.
[32] B.D. Priyaa, M.I. Devi, Fragmented query parse tree based SQL injection detection system for web applications, in: International Conference on Computing Technologies & Intelligent Data Engineering, 2016.
[33] F. Ardjani, K. Sadouni, M. Benyettou, Optimization of SVM multiclass by particle swarm (PSO-SVM), in: International Workshop on Database Technology & Applications, 2010.
[34] SQL injection datasets. Available: https://github.com/omurugur/Sql_Injection_Payload, https://github.com/payloadbox/sql-injection-payload-list, https://www.unb.ca/cic/datasets/url-2016.html.