A SQL Blind Injection Method Based on Gated Recurrent Neural Network

Jiahui Zheng
Cyberspace Institute of Advanced Technology Guangzhou University
Guangzhou, China
2112006052@e.gzhu.edu.cn

Junjian Li
Cyberspace Institute of Advanced Technology Guangzhou University
Guangzhou, China
2112106027@e.gzhu.edu.cn

Chao Li*
Cyberspace Institute of Advanced Technology Guangzhou University
Guangzhou, China
lichao@gzhu.edu.cn

Ran Li
Cyberspace Institute of Advanced Technology Guangzhou University
Guangzhou, China
liran@gzhu.edu.cn

***Abstract*—Security is undoubtedly the most serious problem for Web applications, and SQL injection (SQLi) attacks are one of the most damaging. The detection of SQL blind injection vulnerability is very important, but unfortunately, it is not fast enough. This is because time-based SQL blind injection lacks web page feedback, so the delay function can only be set artificially to judge whether the injection is successful by observing the response time of the page. However, brute force cracking and binary search methods used in injection require more web requests, resulting in a long time to obtain database information in SQL blind injection. In this paper, a gated recurrent neural network-based SQL blind injection technology is proposed to generate the predictive characters in SQL blind injection. By using the neural language model based on deep learning and character sequence prediction, the method proposed in this paper can learn the regularity of common database information, so that it can predict the next possible character according to the currently obtained database information, and sort it according to probability. In this paper, the training model is evaluated, and experiments are carried out on the shooting range to compare the method used in this paper with sqlmap (the most advanced sqli test automation tool at present). The experimental results show that the method used in this paper is more effective and significant than sqlmap in time-based SQL blind injection. It can obtain the database information of the target site through fewer requests, and run faster.**

Keywords—WEB security, Blind SQL injection, Gated recurrent neural network

I. Introduction

With the continuous development of web technology, especially the gradual popularization of 5g technology, the era of the Internet of Things has arrived. The convenience of the network makes the flow of data more timely and valuable. Various web applications not only facilitate people's daily life, and promote social progress, but also bring huge economic benefits. Therefore, many criminals make profits by attacking other people's web applications. SQL (Structured Query Language) injection vulnerability, as the first vulnerability in the report published by OWASP (open web application security project), is the most frequently attacked direction of criminals, which brings a huge security threat. SQL is a structured query language for operating database data. SQL will be used when the application data of web pages interact with the data in the background database. SQL injection is to modify and splice the original URL, form field, or data package input parameters of the web page into SQL statements, which are passed to the web server, and then to the database server to execute database commands. For example, if the developer of a web application directly transmits the data or cookies entered by the user to the database without filtering or verifying (i.e., there is an injection point), it may cause the spliced SQL to be executed, obtain the information and rights to the database, and cause SQL injection attacks.

SQL injection vulnerabilities are classified according to different standards. According to the classification of injection methods, there are mainly the following categories:

- union query SQL injection
- error-based SQL injection
- boolean-based blind SQL injection
- time-based blind SQL injection
- stacked queries SQL injection

Among them, time-based blind injection is also called delayed injection. If a page has neither echoed database content nor reported error information nor Boolean status, we can consider using delay injection. Delay injection is to take the timeline of the page as the judgment basis and inject the information of the database bit by bit. For example, this SQL injection statement, ?id=1' and if(ascii(substr(database(),1,1))= 115,sleep(5),0) --+, does the work of guessing the database name of the target site. 115 refers to ASCII code, and the letter corresponding to 115 is m. if the first letter of the database name of the target site is m, according to the rules of the if statement, this SQL injection will be suspended for 5 seconds, so the tester can judge whether the blind injection based on time is successful according to the page response rate of the target site. For other letters of the database name of the target site, testers usually use the methods of violent guessing and binary search for SQL injection.

Violent guessing means that each letter is compared with each database name letter of the target site in turn. If the guess is successful, the guessing of the next letter will be carried out.

The binary search takes 26 English letters as a table arranged in ascending order, records the middle position of the table, and

* Corresponding author: Chao Li

compares it with the searched keywords. If the two are equal, the search is successful; Otherwise, the middle position is used to divide the table into the first and second subtables. If the keyword recorded in the middle position is greater than the search keyword, the previous sub-table will be further searched. Otherwise, the next sub-table will be searched. Repeat the above process until a record that meets the conditions is found, and the search succeeds, or until the sub-table does not exist, and the search fails. SQL time blind annotation based on the binary search is to guess each letter of the database name of the target site by using the above binary idea.

The above two methods are widely used in many existing SQL injection vulnerability scanning tools. Many security testers also use the above two technologies to write test scripts, and have achieved good results. However, the existing technology has the following shortcomings:

1) For database names, table names, and field names, program developers will not choose meaningless letters to combine when naming these contents but will deliberately choose some meaningful words. For example, admin, username, password, etc. may also be added with the company name and other iconic words. The existing widely used violent guessing and binary search ignore the association between the front and back letters in this database information when conducting blind injection based on time.

2) On the other hand, to distinguish the injection success page from the injection failure page, the time-based blind injection usually selects the sleep statement to pause the target database, so that the page of the target site has a sufficiently obvious pause. To ensure the judgment rate of the success of SQL blind injection, the tester usually sets the pause time of the sleep statement to be a little longer. In this case, the above two methods expose the problem that the test time is too long.

In recent years, with the continuous development of science and technology, machine learning and deep learning are more and more widely used in the field of network security, and have made great contributions to safeguarding the security of cyberspace in the increasingly complex network environment.

In the direction of natural language processing (NLP), a recurrent neural network (RNN) has been widely used because it can process timing data well. To solve the problem of long-term dependence in RNN, LSTM (short-term and long-term memory network model) and GRU (gated cyclic neural network model) have been published one after another. GRU is a variant of LSTM, and LSTM is an improved RNN algorithm, which reduces the gradient vanishing problem of RNN and has been applied on many occasions. The GRU algorithm is to simplify the LSTM model from many aspects. Integrates the input gate and the forgetting gate to form an independent update gate model. At the same time, GRU combines hidden and cell state information to make the final results more efficient than the general LSTM. In the comparison between GRU and LSTM, in fact, generally speaking, the effects of the two models are not very different, but GRU has one gate less than LSTM, which reduces a considerable part of matrix multiplication. In the environment of big data, the GRU model can save a lot of time and improve a large part of efficiency, and its advantages are very obvious.

SQL blind injection technology can learn from the research results of natural language processing and use GRU (gated cyclic neural network model) to process the table names and column names commonly used in the database, so as to optimize the existing SQL blind injection technology, improve work efficiency, and reduce time cost.

The main contributions of this paper are as follows:

1) Through the characteristics of a recurrent neural network suitable for processing time series data, we used the database table name data set and the database field name data set to calculate the internal relevance of common data, trained a model that conforms to the regularity of the database information, and tested the performance of the model.

2) We used the trained model that conforms to the regularity of database information to conducted a time-based blind SQL injection test. By comparing with the traditional method, we reduced the number of guesses in the process of blind SQL injection and reduced the time cost.

II. Related Work

In order to detect and prevent SQL injection attacks, people have proposed different methods in recent ten years, including abnormal SQL query matching, static analysis, dynamic analysis, and so on, aiming to parse or generate sqli statements according to specific SQL syntax.

For example, halfond and orso[7] proposed an AMNESIA that combines dynamic and static analysis methods. In the static analysis phase, the legitimate query model that the application can generate is automatically built. During the dynamic analysis phase, AMNESIA uses runtime monitoring to check whether dynamically generated queries match the model. Mao et al. [8] proposed an intention-oriented detection method, which converts SQL statements into finite deterministic automata and detects SQL statements to determine whether the statements contain attacks.

Biofuzz [9] is a search-based tool that uses the fitness function of context-free syntax to generate test cases. However, because the fitness function is designed manually based on prior knowledge, it is difficult to capture all possible semantic knowledge of sqli attacks. µ4sqli [10] is an automated testing tool. It uses mutation operators to modify test cases, hoping to find more sqli vulnerabilities. However, these mutation operators are designed as fixed patterns, so it is difficult to generate new attacks that are not captured by patterns. To solve the problems exposed by the above SQL injection test case generation method, Muyang et al. [5] proposed a deep natural language processing tool called deepsqli to generate test cases for detecting sqli vulnerabilities. By adopting the neural language model based on deep learning and word sequence prediction, deepsqli has the ability to learn the semantic knowledge embedded in sqli attacks, enabling it to transform user input (or test cases) into a new test case that is semantically related and possibly more complex.

At the same time, the method of combining machine learning with injection-based vulnerability defense is becoming more and more popular. Among them, Kim et al. [14] used the internal query tree from the log to train SVM to classify whether the input is malicious. Sheykhkanloo et al. [15] used the assignment vector attack to train the neural network to classify sqli attacks. Appelt et al. [10] proposed the ML-driven method, which uses context-free syntax to generate test cases and trains the random forest to detect sqli vulnerabilities when the software is running. Jaroslaw et al. [16] applied a neural network to detect sqli attacks, learned normal input, and predicted whether the user's next input was malicious by establishing a model.

This research work has conducted in-depth and detailed research on generating SQL test cases. In terms of detecting SQL injection, good research progress has been made by combining the research results of machine learning and in-depth learning. However, the above method of generating SQL test cases does not conduct in-depth research on time-based SQL blind injection, but only generates available SQL injection test cases, The specific SQL blind injection test still uses the violent guessing method or the binary search method. This paper absorbs the previous research experience, combines deep learning with SQL blind annotation, and optimizes the existing SQL blind annotation technology by training the corresponding language model.

III. SQL Blind Injection Technology Based on Gated Recurrent Neural Network

In order to solve the problem that the traditional SQL blind injection detection method is inefficient, this paper collects the commonly used database table names and column names, tokenizes and samples the data set, and sends it to the designed GRU (gated recurrent neural network model) for training to obtain the model. Finally, it uses the combination of binary search and GRU (gated recurrent neural network model) to perform time-based blind injection detection on the SQL injection site. Finally, the database table name and column name are quickly guessed. The overall algorithm is shown in algorithm 1.

algorithm **1**: **SQL blind injection algorithm based on recurrent neural network**

```
inputs: Lexical element table vocab, Prediction function predict;
outputs: Target database information outputs;
begin
Binary find first character; // Binary find first character
update outputs;// Update target database information
if SQL injection has not been completed then
        for char in predict(outputs) then
    if vocab[char] == ord (next char) then
        update outputs;// Predict characters and update output
    else
        continue guessing next character;
    end if
  Binary find next char;// Change the dichotomy when the number reaches the upper limit
  update outputs;
else
        return outputs;// Database information returned after injection
end if
end
```

For a target site, first, find out the first letter of the required database information according to the binary search method, then select the candidate character list of probability sorting through the GRU model, and inject it in turn. If the prediction times reach the upper limit, use the binary search method. If the injection is successful, judge whether the guess is completed. If the guess is completed, the process ends. Otherwise, continue to guess the next digit. Note that this paper adds the traditional binary search to the recurrent neural network for auxiliary guessing. This is mainly to avoid the failure of the GRU language model due to some rare words. When the GRU model guesses the same letter too many times, it will carry out the binary search to ensure that the correct content is guessed. The binary search algorithm involved in this paper is shown in Algorithm 2.

algorithm **2**: **Binary search algorithm**

```
inputs: Candidate character list vocab_list, Target character target;
outputs: Next character output;
begin
update low, high; // Calculated from candidate character list
calculate mid; // Calculate the intermediate value according to the lower and upper limit values
if vocab_ list[mid] < ord(target) then
        low = mid + 1;
else if vocab_ list[mid] > ord(target) then
        high = mid - 1;
else
    update output;
end if
end
```

The binary search method in SQL injection usually uses ASCII code to compare with the target character. Therefore, it is necessary to sort the candidate characters according to ASCII code to get the sorted candidate character list. The ASCII code range of lowercase English letters a to Z is 141 to 172, the ASCII code range of uppercase English letters a to Z is 101 to 132, and the ASCII code difference of upper and lower case letters is fixed at 40, Therefore, the candidate character list may not contain upper case letters. The initial low and high of the binary search can be set according to the range of the candidate character list. Then calculate the mid-value according to low and high and compare the mid-value with the target. If the mid-value is less than the target value, set low=mid+1 to continue the algorithm. If the mid-value is greater than the target value, set high=mid-1 to continue the algorithm. If the mid-value is equal to the target value, complete the process and return the mid-value.

Before training, the data set text needs to be preprocessed, mainly including the following steps:

- Loads text into memory as a string. In order to reduce the number of lexical elements and facilitate operation, all letters are converted to lowercase letters, and the special characters remain unchanged. At the same time, add * (* does not exist in the selected dataset) at the end of each row to identify the end of each word, so as to prepare for future sampling.

- Splits a string into lexical elements. To predict the database table name and column name, this paper uses a single character as a lexical element.
- Build a dictionary to map the split characters to a numeric index. This paper adopts the frequency sorting method, that is, the smaller the index value, the more frequently the characters appear in the data set, and the index zero represents the unknown characters.
- The text is converted into a numerical index sequence to facilitate model operation.

After preprocessing the data set text, this paper randomly samples the generated dataset digital index sequence, that is, set batch_ size and num_steps for sampling. batch_size represents the batch size, that is, several pieces of data need to be read each time. And num_steps stands for time step, which is mainly selected by the character features in the data set. Since there is no correlation between words in the data set, when collecting feature sets (X sets) and tag sets (Y sets), if the * sign representing the end of a word or phrase appears, start collecting again after the * sign.

In the X and Y sets, each lexical element is represented as a numerical index. It may be difficult to input these indexes directly into the neural network. Therefore, each lexical element is represented as a more expressive feature vector. That is, one hot encoding is used for encoding. In short, each index is mapped to different unit vectors: suppose that the number of different lexical elements in the vocabulary is n, and the range of lexical element indexes is 0 to n-1. If the index of a lexical element is integer i, we will create a vector of all zeros with a length of N and set the element at i to 1. This vector is a unique vector of the original morpheme. For example, if the vector length is 3, then 1 and 2 can be expressed as [0,1,0] and [0,0,1].

The recurrent neural network model used in this paper is GRU (gated recurrent neural network model). The key difference between the gated recurrent unit and ordinary recurrent neural network is that the former supports implicit state gating. This means that the model has a special mechanism to determine when the implicit state should be updated and when the implicit state should be reset. These mechanisms are learnable and can solve the problems listed above. For example, if the first lexical element is very important, the model will learn not to update the implicit state after the first observation. Similarly, the model can learn to skip irrelevant temporary observations. Finally, the model will also learn to reset the implicit state when necessary. The following article will discuss various gate controls in detail.

1) update gate and reset gate

In this paper, they are designed as vectors in the (0,1) interval, so that a convex combination can be performed. Reset gate allows you to control the number of past states that you "may also want to remember"; update gate will allow you to control how many copies of the old state are in the new state.

This paper begins with the construction of these gates. Fig. 1 depicts the input of the reset gate and update gate in the gate control recurrent unit,

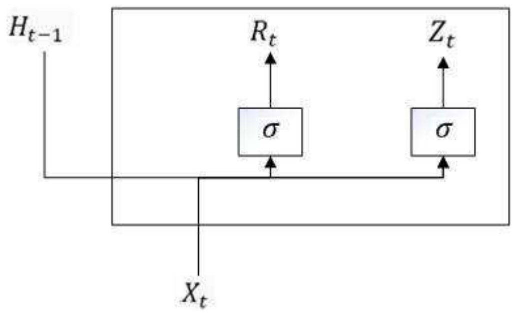


Figure 1 Calculation of reset gate and update gate in gate control recurrent unit model

The input is given by the input of the current time step and the implicit state of the previous time step. The output of the two gates is given by the two fully connected layers using the sigmoid activation function. For a given time step T, assume that the input is a small batch $X_t\langle R^{n*d}$ (number of samples *n*, input number *d*), the implicit state of the previous time step is $H_{t-1}\langle R^{n*h}$ (number of hidden units: h), then reset gate $R_t\langle R^{n*h}$ and update gate $Z_t\langle R^{n*h}$ is calculated as shown in equations (1),

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$
$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)\,, \qquad (1)$$

Where W_{xr} , $W_{xz}\langle R^{d*h}$, W_{hr} , $X_{hr}\langle R^{h*h}$ is the weight parameter and b_r , $b_z\langle R^{1*h}$ is the offset parameter. The broadcast mechanism will be triggered during the summation process. In this article, the sigmoid function is used to convert the input value to the interval (0, 1).

2) candidate hidden state

This paper integrates reset gate R_t and the conventional implicit state update mechanism, based on which the candidate hidden state $\sim H_t\langle R^{n*h}$ in the time step is obtained, as shown in equation (2),

$$\sim H_t = tanh(X_t W_{xh} + (R_t \text{ o } H_{t-1}) W_{hh} + b_h), \qquad (2)$$

Where $W_{xh}\ \langle R^{d*h}$, $W_{hh}\ \langle R^{h*h}$ and are weight parameters, $b_h\langle R^{1*h}$ is an offset term, and the symbol ◦ is a Hadamard product (by element product) operator. In this paper, tanh nonlinear activation function is used to ensure that the values in the candidate implicit states remain in the interval (-1, 1).

The specific process is shown in Figure 2.

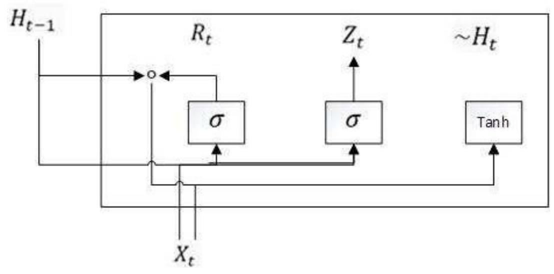


Figure 2 Calculation of candidate hidden states in the gating recurrent unit model

3) hidden state

The above calculation results are only candidate hidden states that need to be combined with the effect of the update gate Z_t. This step determines the new hidden state $H_t \langle R^{n*h}$ to what extent is it from the old state H_{t-1} and the new candidate status $\sim H_t$. The update gate Z_t only needs a convex combination of elements between H_{t-1} and $\sim H_t$ to achieve this goal. This leads to the final update equations of the gate control recurrent unit, as shown in equation (3):

$$H_t = Z_t \circ H_{t-1} + (1 - Z_t) \circ \sim H_t, \tag{3}$$

Whenever the update gate Z_t approaches 1, the model tends to keep only the old state. At this time, the information from X_t is basically ignored, thus effectively skipping the time step t in the dependency chain. On the contrary, when Z_t approaches 0, the new hidden state H_t will approach the candidate hidden state $\sim H_t$. These designs can deal with the gradient vanishing problem in recurrent neural networks and better capture the dependencies of sequences with long time steps. For example, if the update gates of all time steps of the whole subsequence are close to 1, the old hidden state at the start time step of the sequence will be easily retained and transferred to the end of the sequence regardless of the length of the sequence.

The specific calculation flow is shown in Figure 3.

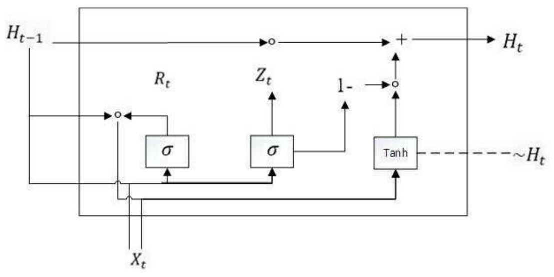


Figure 3 Calculation of hidden states in the gating recurrent unit model

After training the model, a warm-up method is used for character prediction. Str is a string containing one or more characters. When we loop through the start character in str, we constantly pass the implicit state to the next time step, but do not generate any output. Because during this period, the model will update itself (for example, update the hidden state), but will not predict. After the warm-up, the value of the hidden state is usually more suitable for prediction than the initial value at the beginning, so as to predict the characters and output them. Since the characters are uniquely encoded in this paper, SoftMax can be used to calculate the probability of each position on the output vector. The subscript corresponding to the probability in the output vector is the numerical index of the corresponding character in the lexical element table. In this way, the probability distribution of the next character corresponding to the output string can be obtained. When predicting characters, take the characters with the highest probability from the probability distribution list until the number of errors reaches the upper limit, and then use the binary search method to guess.

IV. Experimental Result

The data set used in the experiment comes from the database common table name text and column name text contained in sqlmap. Sqlmap usually uses these texts for brute-force cracking. This paper trains the recurrent neural network language model and predicts the characters on the basis of these texts. Some data are shown in Table 1,

Table 1 partial database table names and column names

Database table name	Database column name
users	name
customer	user_id
orders	description
employee	username
Category	group_id
Customers	first_name
Country	itemid
config	category_id

Read the database general table name text and column name text respectively. The general table name text has 3558 table names, 43209 lexical elements, 38 alphanumeric characters, and special characters. 3202 table names are randomly selected for training, with a total of 39239 lexical elements. The table name test set contains 356 table names and 3970 lexical elements.

There are 2733 column names, 25631 lexical elements, 37 alphanumeric characters and, special characters in the general column name text. 2450 column names are randomly selected for training, with a total of 23329 lexical elements. The column name test set contains 283 column names and 2302 lexical elements.

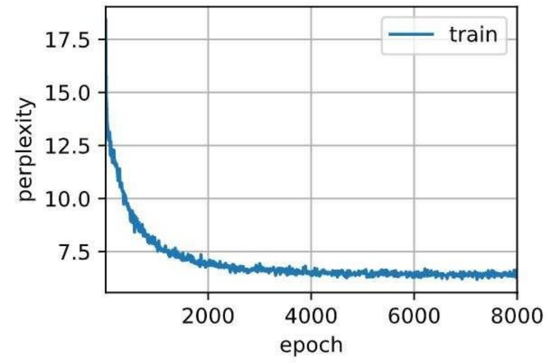
The training parameters of the model are as follows: Set the number of hidden units to 512, the batch size to 8, the time step to 3, the training times to 8000, and the learning rate to 0.1.

The quality of the language model is evaluated by the degree of confusion. See equation (4) for the calculation of traditional confusion.

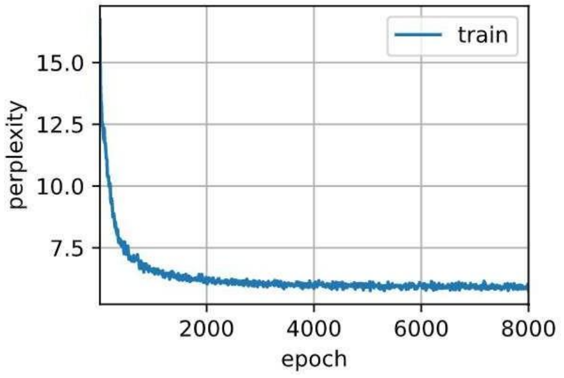
$$Exp\left(-\frac{1}{n}\sum_{t=1}^{n} log\, P\left(x_t \mid x_{t-1}, \ldots, x_1\right)\right) \tag{4}$$

Where p is given by the language model, X_t is the actual lexical element observed from the sequence in time step t. the brackets represent the average value of cross-entropy loss of all n lexical elements in a sequence. For historical reasons, scientists in natural language processing prefer to perform exponential operations. In this article, nn.Crossentropyloss() functions to calculate the cross-entropy and then calculate the degree of confusion.

The calculation of the degree of confusion of the database table name and the database column name dataset is shown in Figure 4,



(a) Confusion degree of database table name dataset

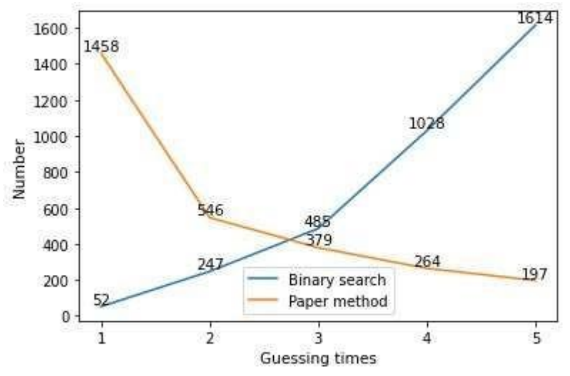


(b) Confusion curve of the data set of the database column name.

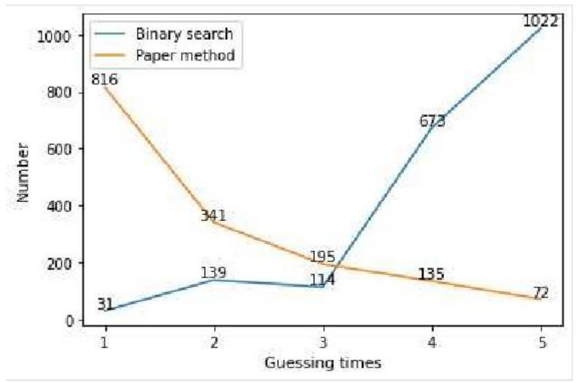
Figure 4 data set confusion curve

It can be clearly seen from Figure 4 that after 8000 iterations, the degree of confusion of the database table name data set is 5.8, and the training speed is 3589.8 words per second. The degree of confusion of the database column name data set is 6.7, and the training speed is 4055.0 words per second.

More tests are needed to evaluate the quality of the model. Therefore, the binary search method and the method described in this paper are used to predict character for the two test sets. The results are shown in Figure 5,



(a) Cross-reference table of the database table name test set result



(b) Cross-reference table of the database column name test set result

Figure 5 comparison table of test set results

The final number of lexical elements in the two methods in Figure 5 is not equal. This is mainly because the method in this paper uses the dichotomy to guess the first character of each table name or column name, which is not included in the statistics. The number of lexical elements is not equal to the size of the test set. This is mainly because some characters have been predicted more than five times, while Figure 5 only records the number of successful predictions within five times.

In Figure 5 it can be found that the number of successful binary search methods increases with the number of guesses, mainly after four guesses. The number of successful binary search methods decreases with the number of guesses, mainly in the first two guesses. The model trained in this paper can successfully predict 50.5% of database table name characters and 50.3% of database column names within two times. Based on these data, this paper sets the upper limit of guessing failure of the language model to 2 in the next experiment, that is, if the guessing fails twice, it will switch to the binary search mode.

The SQL blind injection test statement is as follows,

Get the number of tables,

```
1' and (select count(*) from information_schema.tables
where table_schema=database())>?#
```

Get the length of each table,

```
1' and (select length(table_name) from
information_schema.tables where table_schema=database()
limit 0,1)>?#
```

Guess each table name one by one,

```
1' and (ascii(substr((select table_name from
information_schema.tables where table_schema=database()
limit 0,1),0,1)))=?#
```

The same is true for the SQL blind annotation statement for obtaining column names.

Sqlmap is an SQL injection tool widely used in academia and industry [9-13]. This article will compare it with sqlmap on the same SQL blind injection range site. The server used in this experiment is apache2.4.39, the back-end language is php7.3.4nts, the database version is mysql8.7.2, and the widely used sqllab shooting range is used in the shooting range. The database table names and column names are from the test set.

The final results are shown in Table 2 and Table 3, respectively.

Table 2 Comparison of request times

Site name	sqlmap	Paper Method
Site I	1556	322
Site II	2661	1023
Site III	2891	1108

Table 3 Second consumption comparison table

Site name	sqlmap	Paper Method
Site I	712	296
Site II	1689	912
Site III	1657	709

From the results, it can be found that the method used in this paper has fewer requests and takes less time than the binary search method used by common tools such as sqlmap. However, since the page response time of sqlmap is dynamically set and the page response time of this paper is statically set to a constant, the single request takes longer than sqlmap. Even so, SQL blind injection technology based on a gated recurrent neural network proposed in this paper still achieved good results.

V. CONCLUSION AND FUTURE WORK

In view of the increasingly serious SQL injection vulnerability, the traditional time-based SQL blind injection method has exposed its shortcomings of taking too long. Based on existing technology and research results in the field of natural language processing, this paper proposes a SQL blind injection technology based on gated recurrent neural network, which combines traditional binary search technology with the idea of the recurrent neural network and has achieved remarkable results. In the future, excellent algorithms such as recurrent neural networks can be combined with other SQL injection problems and even other web security problems to provide better solutions to resist the increasingly serious network threats.

ACKNOWLEDGMENT

This work was supported by the Major Research Plan of the National Natural Science Foundation of China (92167203), the National Natural Science Foundation of China (61872100), the China Postdoctoral Science Foundation (2022M710860), the Guangzhou Science and Technology Plan Project (202102010440).

REFERENCES

[1] Zhuo Z , Cai T , Zhang X , et al. Long short − term memory on abstract syntax tree for SQL injection detection[J]. IET Software, 2021(3).

[2] Marashdeh Z , Suwais K , Alia M . A Survey on SQL Injection Attack: Detection and Challenges[C]// 2021 International Conference on Information Technology (ICIT). 2021.

[3] Zhang G, Zhang Y, Wang Y, et al. A fine-grained petri model for SQL time-blind injection[C]//2021 International Conference on Networking and Network Applications (NaNA). IEEE, 2021: 161-167.

[4] Hu J , Zhao W , Cui Y . A Survey on SQL Injection Attacks, Detection and Prevention[C]// ICMLC 2020: 2020 12th International Conference on Machine Learning and Computing. 2020.

[5] Liu M, Li K, Chen T. DeepSQLi: Deep semantic learning for testing SQL injection[C]//Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2020: 286-297.

[6] Anagandula K, Zavarsky P. An analysis of effectiveness of black-box web application scanners in detection of stored SQL injection and stored XSS vulnerabilities[C]//2020 3rd International Conference on Data Intelligence and Security (ICDIS). IEEE, 2020: 40-48.

[7] William G. J. Halfond and Alessandro Orso. 2005. AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks. In ASE'05: Proc. of the 20th IEEE/ACM International Conference on Automated Software Engineering. 174–183.

[8] Chenyu M, Fan G. Defending SQL injection attacks based-on intention-oriented detection[C]//2016 11th International Conference on Computer Science & Education (ICCSE). IEEE, 2016: 939-944.

[9] Thomé J, Gorla A, Zeller A. Search-based security testing of web applications[C]//Proceedings of the 7th International Workshop on Search-Based Software Testing. 2014: 5-14.

[10] Appelt D, Nguyen C D, Briand L C, et al. Automated testing for SQL injection vulnerabilities: an input mutation approach[C]//Proceedings of the 2014 International Symposium on Software Testing and Analysis. 2014: 259-269.

[11] Appelt D, Alshahwan N, Briand L. Assessing the impact of firewalls and database proxies on sql injection testing[C]//International Workshop on Future Internet Testing. Springer, Cham, 2013: 32-47.

[12] Appelt D, Nguyen C D, Panichella A, et al. A machine-learning-driven evolutionary approach for testing web application firewalls[J]. IEEE Transactions on Reliability, 2018, 67(3): 733-757.

[13] Sinha S. Sql mapping[M]//Beginning Ethical Hacking with Kali Linux. Apress, Berkeley, CA, 2018: 221-258.

[14] Kim M Y, Lee D H. Data-mining based SQL injection attack detection using internal query trees[J]. Expert Systems with Applications, 2014, 41(11): 5416-5430.

[15] Sheykhkanloo N M. A learning-based neural network model for the detection and classification of SQL injection attacks[J]. International Journal of Cyber Warfare and Terrorism (IJCWT), 2017, 7(2): 16-41.

[16] Skaruz J, Seredynski F. Recurrent neural networks towards detection of SQL attacks[C]//2007 IEEE International Parallel and Distributed Processing Symposium. IEEE, 2007: 1-8.