IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 22, 2018, accepted November 16, 2018, date of publication November 20, 2018,
date of current version December 27, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2882517

Webshell Traffic Detection With Character-Level Features Based on Deep Learning

HUA ZHANG[1], HONGCHAO GUAN[1], HANBING YAN[2], WENMIN LI[1], YUQI YU[3], HAO ZHOU[2], AND XINGYU ZENG[1]

[1]State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
[2]National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing, China
[3]School of Economics and Management, Beihang University, Beijing, China

Corresponding authors: Hua Zhang (zhanghua_288@bupt.edu.cn), Hanbing Yan (yhb@cert.org.cn), and Wenmin Li (liwenmin02@outlook.com)

This work was supported by the National Natural Science Foundation of China under Grant U1736218 and under Grant 61502044.

ABSTRACT Webshell is a kind of backdoor programs based on Web services. Network-based detection could monitor the request and response traffic to find abnormal behaviors and detect the existence of Webshell. Some machine learning and deep learning methods have been used in this field, but the current methods need to be further explored in discovering new attacks and performance. In order to detect large-scale unknown Webshell events, we propose a Webshell traffic detection model combining the characteristics of convolutional neural network and long short-term memory network. At the same time, we propose a character-level traffic content feature transformation method. We apply the method in our proposed model and evaluate our approach on a Webshell detection testbed. The experiment result indicates that the model has a high precision rate and recall rate, and the generalization ability can be guaranteed.

INDEX TERMS Webshell, character level, convolutional neural network, long short term memory.

I. INTRODUCTION

In recent years, a series of network security incidents have been widely concerned, most of which are closely related to the security of web website, such as Struts2 Leak and Prism Gate. According to the annual safety report issued by National Computer Network Emergency Response Technical Team/Coordination Center of China (CNCERT/ CC), the invasion behavior related to website is drastically increasing every year [1]. Among various security threats, the backdoor of the website is extremely serious. The backdoor of the website is also called Webshell, which is a backdoor program based on web services. A webmaster can use the webpage to upload files, view the database, and execute OS commands through the browser. At the same time, malicious user can also initiate attacks by the Webshell tools such as China Chopper.

Malicious Webshell detection methods can be divided into two types: host based detection and network based detection [2]. The former includes regularization feature matching approach, statistical feature thresholding approach (such as NeoPI [3]). This kind of detection method is intuitive, which make it is easy to be circumvented by attackers. The network based detection methods mainly use communication traffic to detect Webshell attacks. The traditional network-layer-only security controls such as firewalls and signature-based intrusion prevention and detection systems have little role to play in detecting Webshell. Webshell as a script file has several static features and dynamic features.

Contribution: The main motivation of this paper is to discover Webshell attacks, especially unknown attacks, from network traffic. In order to achieve this goal, we consider the advantage of deep learning in generalization ability. We propose a deep learning model architecture combining Convolutional Neural Network (CNN) [4], [5] with Long Short Term Memory (LSTM) [6], [7]. CNN is applied to extract local key field features, and the features of text sequences are captured by LSTM. The combination of these two methods can mine patterns of Webshell malicious traffic. The main contributions of this paper can be summarized as follows:

- We propose the character level method to transform Webshell content feature, which can completely retain the sequential pattern characteristics of traffic content. At the same time, the dimension can be reduced, and the effect of the model detection is improved. In the actual case, there are a better performance in the hundreds of millions of traffic items.
- We propose a deep learning detection model structure combining CNN with LSTM. By splicing

2169-3536 © 2018 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

the LSTM layer into the hidden output results of CNN layers, we can achieve a comprehensive extraction of local features and sequence features. The combined model detection is more effective than the single CNN model and the single LSTM model.

- We evaluate our proposed approach on a testbed, the results shows it has higher precision, higher recall, and higher F1-score. It is feasible in real traffic detection and has the ability to discover the unknown Webshell.

II. DATA PREPARATION

A. SELECTION OF WEBSHELL TYPES

According to the server-side environment, we can mainly divide Webshell into PHP based Webshell, ASP based Webshell and JSP based Webshell, where PHP for Apache, JSP for Tomcat, ASP for IIS. According to the size and function of the scripting program, Webshell can be divided into big Trojan, small Trojan and One Word Trojan. The introduction of these three kinds of Webshell are as follows:

- ***Big Trojan***, a common Webshell with comprehensive function. It usually contains friendly operation interface, which can carry on file operation, command execution and database operation under the graphical interface. The Trojan often has a large file, and the codes are confused to prevent the detection. In addition, some big Trojans will contain a login interface.
- ***Small Trojan***, which contains only one function. A small Trojan usually provides file upload or database lifting. When the website has a limited file size, the attacker will use small Trojan as an upload springboard. The small Trojan file size is often within 5KB, and there is no password protection.
- ***One Word Trojan***, which refers to a script code that is confused or encoded, usually a command execution code, such as the ''eval()'' function. It can be inserted into the original web code.

In this paper, our motivation is to find Webshell attacks from traffic, especially unknown attacks. So when analyzing the traffic, we divide Webshell into two following types.

- ***Browser&Server (B&S) Based Webshell***. This kind of Webshell is mainly used in web-based application. Hackers upload Webshell files to target websites through vulnerabilities such as file upload and establish connections with uploaded web shells through browsers. The browsers use *Post* method to send command parameters, the contents of the returned package are usually the contents of the HTML structure returned after the execution of the control command.
- ***Client&Server (C&S) Based Webshell***. This kind of Webshell is based on the client sending commands to the server. Take the China Chopper as an example, the client encodes the command execution scripts by Base64, and connects the server to a One Word Trojan program, the server uses the specified separator to separate the instructions execution results and returns them to the client. From the traffic point of view, the result of server return is usually the result of command execution, rather than the complete HTML page structure, moreover, the request package contains more execution content.

These two kinds of Webshell pass request information through URL and *Post* body. At the same time, it can contain all kinds of Webshell categories mentioned above, such as ''big Trojan, small Trojan, One Word Trojan''.

B. DATA COLLECTION

The research object of this paper is malicious HTTP traffic of Webshell. We collect massive website communication traffic extracted from university network monitor system, including normal traffic and anomaly traffic. After accumulating for a long time, we obtain about 600,000 Webshell malicious traffic which include *request* and *response* body. At the same time, we simulate the Webshell attack in our lab environment. The server environment includes Apache, Tomcat, IIS and so on. The programming languages include PHP, JSP, and ASP. We use the kali platform [8] to build the test environment that provides Webshell environments such as *Webacoo, Weevely* and so on. The server-side Webshell file type contains big Trojan, small Trojan and One Word Trojan. We use spider tools to obtain these kinds of communication traffic generated by these Webshell tools, including *Post* request and *Get* request. After data processing, these Webshell malicious traffic data and normal traffic data are regarded as training data sets.

C. DATA CLEANING

We perform data cleaning operations on the collected Webshell traffic, mainly including the following operations.

1) URL DECODE

We find that URL data often be encoded. For instance, the URL entry is ''http://www.oschina.net/search? scope=bbs&q=∗&%5E%25$'', after URL decode process, the result is ''http://www.oschina.net/search?scope=bbs&q= ∗ &^%$''. In this way, we can obtain correct URL format.

2) BASE64 DECODE

For safe transmission or prevention detection, many *Post* request packets are encoded by Base64 encode method. For encoding methods, we need first identify the starting position of the encoding and try to decode data with corresponding method. Taking the *China Chopper*client-side request traffic as an example, the command code is generally encoded into Base64 code to confuse traffic information, and the data is decoded by Base64 decode method at the server-side, finally the command is passed to the*One Word Trojan* program.

3) ELIMINATING THE EFFECT OF ENCRYPTION

For encryption traffic, since we cannot carry out the corresponding decryption, we first identify the encrypted data location, and then replace the corresponding part to the predefined flags. It is helpful for reducing the interference to the model prediction results.

4) BINARY DATA STREAM

Considering file uploading, video streaming, and other similar situations, the binary stream data in *Post* request body does not contain semantic information, which is a disturbance factor for model checking. Similar to the above method, we first identify the location of the binary stream and replace the data in the corresponding location with the predefined flags. The replace entries are shown in Table 1.

TABLE 1. Characters replacement rules.

String Type	Replacement string
MD5 [9]	MD5_HASH
SHA1 [10]	SHA_HASH
BASE64	BASE64_ENCODE
Encryption	Encryption
Binary Data Stream	Binary

III. CHARACTER LEVEL CONTENT FEATURE OF WEBSHELL TRAFFIC

We extract the characteristics of URL and *Post* body separately, and finally generate the feature vectors detection model. Meanwhile, we divide these data into three sets: training set, verification set and test set.

A. CHARACTER LEVEL CONTENT FEATURE TRANSFORMATION

For content feature, which includes URL and *Post* body, we take these parts of information as natural language. However, because of most parts of these content cannot be divided into words by space, we need to use word segmentation to quantify text data. Nowadays, word segmentation based on N-Gram [11] is widely used in text categorization. Considering the use of N-Gram segmentation method will generate a huge dimension space of word dictionary, some dimensionality reduction methods are proposed to extract key information, thereby enhancing the effectiveness of the model. Most widely used dimensionality reduction methods are TF-IDF [12] and Word2Vec [13]. The introduction of these methods are described as follows.

1) N-GRAM WITH TF-IDF

Assume N=3, after 3-ram process, we can get the words list contains all words with length 3. To reduce the huge dimension space, we drop useless words, such as ''123'', and retain important words, such as ''exe''. The main idea of TF-IDF [14], [15] is that if a word or phrase has a high term frequency in an article and rarely appears in other articles, it is considered to be important and it is extremely suitable for classification. However, dimension remains at least ten thousands after dimensionality reduction. What's more, the absence of data features exists after dimensionality reduction.

2) N-GRAM WITH Word2Vec

Word2vec [13] was created by a team of researchers led by Tomas Mikolov at Google. The algorithm has been subsequently analyzed and explained by other researchers [16]–[18]. Embedding vectors created using the word2vec algorithm have many advantages compared to earlier algorithms [19] such as latent semantic analysis. After word embedding, each word can be represented as fix length vector, the dimension can be reduced a lot. However, for massive data processing needs, the model needs a lot of time and resources for training and generating word vector [20].

3) OUR CHARACTER LEVEL METHOD

Considering the problem mentioned above, we propose character level [21] content feature transformation method to transform content information to feature vector.

For long traffic content, we define a fixed length L. If the content length is more than L, we truncate the data of L length. If the content length is less than L, we repeat the data in the front part to the end of the data until the data length is equal to L, which is different from the complement 0 mentioned in most of the papers. In contrast, we have proved that the method of intercepting the preceding data and filling can get better model checking results.

For each entry, we transform character sequence into corresponding ASCII value sequence. If encountering non ASCII visible characters, such as Chinese characters, we drop these characters and finally the feature vectors is formed.

For example, the request content is ''/1.qaz.jsp chopper=i&z0=gb2312'', the length of content is 30, the corresponding ASCII vector is [47, 49, 46, 113, 97, 122, 46, 106, 115, 112, 32, 99, 104, 111, 112, 112, 101, 114, 61, 105, 38, 122, 48, 61, 103, 98, 50, 51, 49, 50]. The part of less than L supplemented the previous contents.

B. FEATURE ANALYSIS

For request packets, we splice URL and *Post* body to generate the original data. After analysis the content feature of Webshell traffic, we find that the distinctive features are as follows.

1) URL Feature

Webshell files are often located in sensitive directories such as ''*images/*'', ''*css/*''. These files are often named with sensitive names, such as ''*dama.jsp*'', ''*shell.php*''. The file suffix name is generally php, *asp*, and *jsp*. There also exists confused file suffix such as ''*.asp;png*'' to avoid file upload validation.

2) POST BODY FEATURE

Take the China Chopper as an example. The *Post* body content is like this: ''*c=@eval (base64_decode ($_post[z0]));&z0=*''', we find that some PHP system call command is embedded into *Post* body values after being encoded by *Base64*. What's more, the key command

code is encoded with *Base64*, which is very different from normal traffic. Some examples of Webshell request traffic are shown in Table 2.

TABLE 2. Webshell request traffic example.

URL	POST BODY
/1.qaz.jsp	chopper=i&z0=gb2312&z1=/**/a.jsp& z2=/** /88/b.jsp
/admin/123.jsp;png	passwd=******
/bak.jsp	z0=utf-8&z1=/usr/***/***/**.sh
/dvwa/upload/ex0shell.php	
/wszba/script/../../../etc/passwd.js	
/admin/database/dama.jsp	
/dvwa/test/ak-74.php	action=exesys&cmmd=ls
Webshell/C99madShell.php	

After feature analysis, we determine the length of the fixed length sequence and the specific character replacement rules. The string length distribution of the original data box diagram is shown in the Fig.1. The *"len"* label denotes the length of URL and *Post* body after stitching. The *"urllen"* and *"postlen"* labels denote the length of URL and *Post* body respectively. We can find that the 3/4 score is approximately 300 bytes. That is to say, 75% of data are less than 300 bytes. After the statistic, we define the fixed length of sequence vector equal to 300 bytes. Finally, as the data preparation module mentioned before, we can obtain ASCII sequence vector for each request traffic sequence.

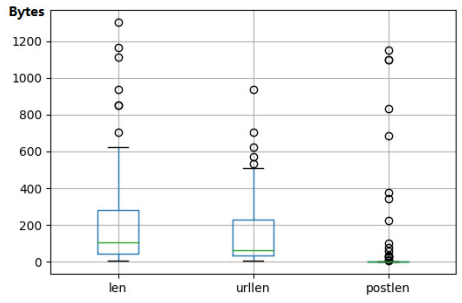

FIGURE 1. Data length distribution.

IV. WEBSHELL DETECTION MODEL BASED ON DEEP LEARNING

The model is based on Convolution Neural Network (CNN) [4], [5] and Long Short-Term Memory Network (LSTM) [6], [7]. The main reason for model selection is that CNN can get local features, and can identify key malicious code fragments. After that, we input the hidden output vector into LSTM layers. In this layers, sequence pattern feature can be learned. The model architecture is shown in Fig.2.

We transform request data into input tensor, feed tensor into then three layers of one dimensional convolution

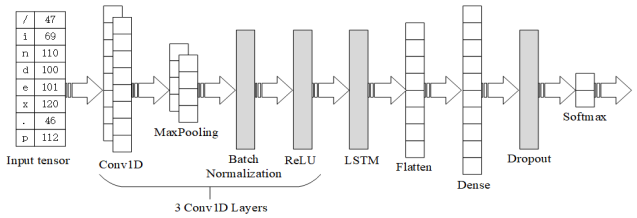

FIGURE 2. Webshell detection model structure.

neural network. Then we feed the hidden output into LSTM layer. After the output flatten, we feed the output vector into the fully connection layer. Finally, the softmax classifier is used to output the two classification results.

It is necessary to note that the actual convolution kernel number and other hyperparameters need to be set according to the actual problem. In this study, we determine that the number of convolution kernels are 24, 48, 64, the LSTM units we set are 64, and the number of fully connected layer neurons is 128. The core layers are described as follows.

3) CONVOLUTION AND ReLU [22]

We suppose the filter, where h is the filter window width. In our research, h means the length of filter words, we define $h = 3$ after experiment contrast. At position i of the input vector X, the length of X is n, the result of convolution is C_i:

$$C_i = f(W \cdot X_{i:i+h-1} + b) \tag{1}$$

Where b is the bias and f is the nonlinear rectify function. The convolution result of the whole sentence is a feature vector, we use the same padding, so the length is n, and the result is C.

$$C = [C_1, C_2, \ldots, C_n] \tag{2}$$

We use the *ReLU* activation function, which is an element-wise operation. This activation function can get better gradient in back-propagation.

4) MAX-POOLING

A max-pooling operation on the feature map (result of convolution) is applied to obtain the maximum value from C_i to C_j, the feature map size in our research is 2, so we can get $C_{i,i+1}$ as the feature corresponding to the particular filter size 2.

$$C_{i,i+1} = max\,(C_i, C_{i+1}) \tag{3}$$

It captures the most important value (the highest value) for each feature map. Besides, this operation can naturally deal with variable sentence lengths. Then, the max-pooling results are concatenated to get a feature vector $Z = [C_{i,i+1}, C_{i+2,i+3}, \ldots, C_{n-1,n}]$.

5) BATCH NORMALIZATION [23]

Traditional neural networks only standardize the X before entering the sample X into the input layer to reduce the difference between the samples. On this basis, batch normalization

not only standardizes the input data X of the input layer, but also standardize the input of each hidden layer. We find that after convolution process, the hidden layer output data are not satisfied with normal distribution. We use batch normalization to improve the model effect. Experimental results in section 5 show that after batch normalization entry, the effect of the model has been significantly improved.

6) LONG SHORT-TERM MEMORY

Recurrent Neural Networks (RNNs) [6], [7], which can exploit information both from the past and the future to improve the prediction performance and learn the complex patterns in HTTP requests better. In our research, we use LSTM to learn the hidden embedding vector pattern generated from Convolution layers. Given embedding vector $\{v_{11}, v_{12}, v_{13}, \ldots, v_{1z}\}$ of content sequence of request R_i, the forward LSTM $\vec{f}$ reads the input sequence from v_{11} to v_{1z}, and calculates a sequence of forward hidden states $(\vec{h}_{11}, \vec{h}_{12}, \ldots, \vec{h}_{1z})(\vec{h}_{1i} \in R^p$ and p is the dimensionality of hidden states). After that, the output vector of the LSTM layer is fed into next layer.

7) DENSE

After flatten layers, the convolution result can be feed into dense layer, the output of the dense layer is D.

$$D = [D_1, D_2, \ldots D_j, \ldots, D_l] \tag{4}$$

$$D_j = f(\sum_{k=0}^{H} W_{kj} * X_k + b_j) \tag{5}$$

The flatten output result with length H, and the dense layer units with count l, W_{kj} represent the weight vector for X_k in j-th dense unit, b_j represent the bias of j-th dense unit. For each unit we can get output alias as D_j, finally we concatenate each units output to get the dense result D.

8) DROPOUT [24]

Before mapping to the binary output, in training process, a dropout layer is appended. Dropout is an important strategy to suppress overfitting problem. It drops the output weights of each hidden units randomly.

9) SOFTMAX

The softmax function can compress a vector of a K dimension containing any real number to another K dimensional vector, in which each element fall in (0, 1), and the sum of all the elements is 1. We use the output of softmax function as our classification layer output. The probability that the sample vector x belongs to the j-th classification is:

$$P(y = j|x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T w_k}} \quad (j = 1, 2, \ldots, K) \tag{6}$$

We get the class label by the argmax function. Finally, we can get the classification result.

V. EVALUATION

A. WEBSHELL TRAFFIC DETECTION TESTBED

We evaluate the models of this paper on a testbed, as shown in Fig.3. We can see that the whole detection process is divided into three stages.

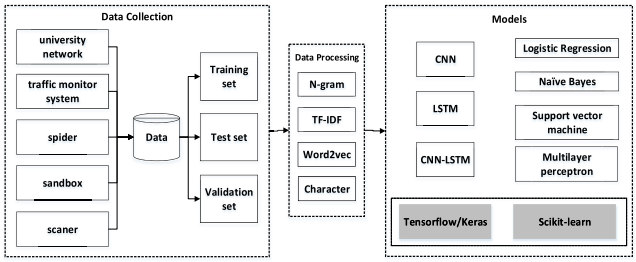

FIGURE 3. Webshell traffic detection testbed.

The first stage is data collection. We adopt a variety of methods to build data sets, including campus network traffic collection, traffic monitoring system to collect traffic, using crawler tools to crawl webshell *request* packets in the test environment, running a variety of types in the sandbox environment, and the traffic of Webshell scanned by the scanner. The data is then divided into training set, test set and validation set.

The second stage is the data preprocessing stage. We adopt 3-gram [11] with TF-IDF [12] method, 3-gram [11] with word2vec [13] method and character-based vectorization method, and output the data preprocessing results to the model for use.

At last, we adopt a multi-model comparative experiments, including several machine learning models and deep learning models. The deep learning model mainly depends on Tensorflow/Keras in the underlying environment and the machine learning depends on Scikit-learn in the underlying environment. The server hardware environment is composed of 256G memory, 8 core E5-2600v2 series, NVIDIA GTX1080TI public version and 14T hard disk. Operating system is Ubuntu Server 16.04 LTS.

B. DATASET

Train Dataset Distribution: For Webshell traffic, we collect data from real traffic monitor system and simulate Webshell attack in local environment. In addition, other Webshell traffic are collected by Webshell vulnerability scans under experimental environment. The normal traffic is captured from HTTP traffic packets via the network gateway. We perform manual verification and tagging for these data. The number of dataset is 949807, including 634969 normal traffic data and 314838 Webshell traffic data. Among these traffic data, 125036 entries adapt *Post* methods, others adapt Get methods. The distribution of dataset is shown in Table 3. It should be noted that data type represents the different behaviors of Webshell and different types of Webshell. Such as ''scan shell file'' response to the scanning behavior for requesting a

TABLE 3. Type of webshell traffic dataset.

Data type	Count
China Chopper	10087
One Word Trojan	27024
Scan Shell File	49701
Execute Shell Command	45652
Other Types	182374

shell file. Other types response to Webshell data beyond the scope of the rule recognition.

According to the Fig.3, we divide the dataset into three parts, including training set, test set and validation set. The corresponding ratio are 60%, 20% and 20%.

C. MODEL COMPARISON

We compare model effect with different models. The number of test dataset is 189965, including 126707 normal and 63258 Webshell traffic, the result is shown in Table 4.

TABLE 4. Result comparison.

Model	Precision	Recall	F1-score
3-gram_TF-IDF_LR	0.9960	0.6531	0.7889
3-gram_TF-IDF_NaiveBayes	0.4706	0.9798	0.6358
3-gram_TF-IDF_SVM	0.8348	0.7854	0.8093
3-gram_TF-IDF_MLP	0.9931	0.6575	0.7912
3-gram_Word2Vec_CNN	0.9689	0.9623	0.9656
3-gram_Word2Vec_LSTM	0.9541	0.9713	0.9626
3-gram_Word2Vec_CNN_LSTM	0.9782	0.9645	0.9713
Character_CNN	0.9852	0.9694	0.9773
Character_LSTM	0.9748	0.9759	0.9753
Character_CNN_LSTM	0.9828	0.9784	0.9851

Firstly, we compare the model effects between machine learning methods, including Logistic Regression, Naive Bayes, Support Vector Machine, and Multilayer perceptron. We use the participle method of 3-grams and extract the first 100 thousand words according to TF-IDF to generate the word vocabulary. The results show that machine learning methods do not perform well on F1-score indicators. In other words, machine learning methods cannot achieve the equilibrium of precision and recall.

Secondly, we compare the performance of deep learning methods including CNN, LSTM and CNN_LSTM. We summarize that no matter what embedding method applied, including the word2vec based feature extraction method and the character level feature extraction method, the combination method CNN_LSTM has a better performance on the test set, has higher F1-score than other methods. Additionally, the character level feature transform methods has a positive influence on model accuracy.

Especially, we compare the speed of convergence in training stage between three kinds of character level deep learning models. The result is shown in Fig.4, we train three models in 5 epochs and record the accuracy and loss values of each

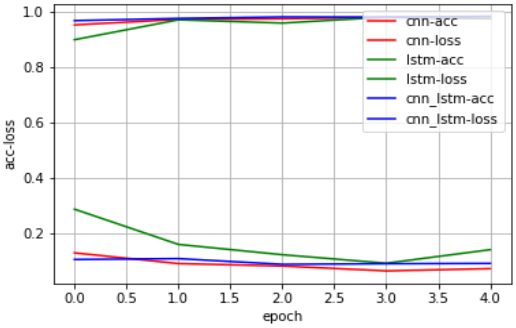

FIGURE 4. Model convergence speed comparison.

epoch in validate set. The results show that the CNN_LSTM model has the fastest speed of convergence compared with CNN model and LSTM model.

D. PARAMETER TUNING

In the process of optimization of the proposed model, we adjust the values of different hyperparameters and carry out comparative experiments in order to determine the corresponding values of each hyperparameter under the optimal effect of the model.

1) CONTENT PADDING TYPE

We contrast our proposed text padding method with zero padding method to determine the optimal filling method. The text padding means that for text whose length is less than the specified length, we intercept the front part of the text and add it to the back. The detection is based on Character_CNN_LSTM model. The different performances of the two methods on the test set are shown in Fig.5.

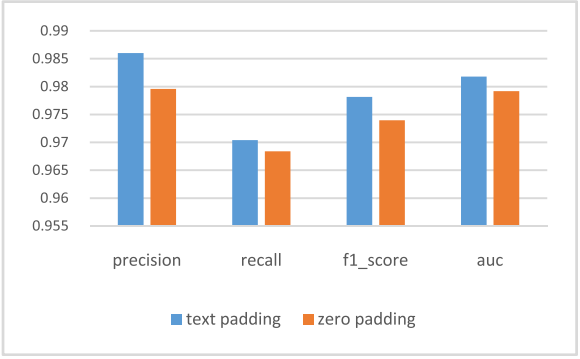

FIGURE 5. Model performance on padding methods.

What needs to be pointed out is the AUC means the area under the receiver operating characteristic curve. We can find that text padding perform better than zero padding in each evaluation standard.

2) DATA LENGTH

We contrast our model effects with different input data length, the optional length are 200 bytes, 300 bytes, 400 bytes,

the comparison result is shown in Fig.6. As shown in the figure, when the length is set to 300 bytes, the value of recall, F1-score and AUC is higher.

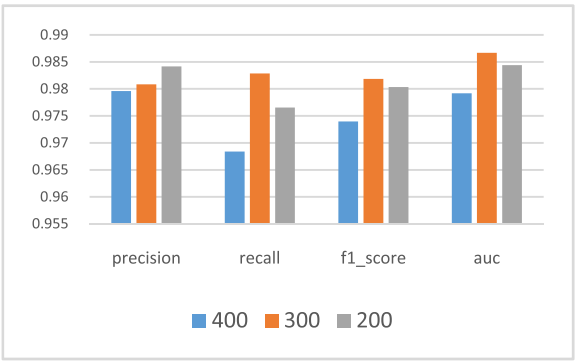

FIGURE 6. Model performance on data length.

3) BATCH NORMALIZATION

We adopt the idea of Batch Normalization in hidden layer output. In our proposed model, we add Batch Normalization layer after the Max-Pooling layer. The model performance result is shown in Fig.7. We find that the Batch Normalization play an important role on the effect of the model. In particular, with batch-normalization, the recall of the model has been greatly improved. This shows that the model has a stronger ability to detect abnormalities.

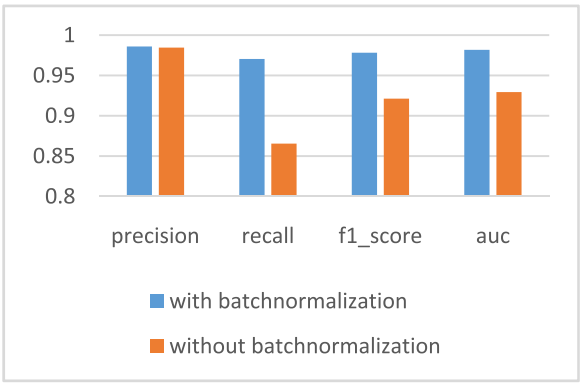

FIGURE 7. Model performance on batch normalization.

E. DETECTION SPEED CONTRAST

Considering the huge volume of traffic data, the detection speed is also the key factor of the feasibility of the model in the real environment. We compared the difference of training and detection speed between the deep learning model and the machine learning model through experiments. We extract 200,000,000 traffic entries of from the traffic testbed. The size of the traffic data is 2.86T. The training dataset is shown in Table 3. Each model uses the same test environment of software and hardware. The environment information is as described above, we test the time required for each model to complete all training and detection task, and the results are shown in Fig.8. and Fig.9. We can find that the character-based deep learning model has faster training and prediction speed than word2vec-based deep learning model, especially the advantage of speed in training is more obvious. The main reason is that word2vec-based deep learning model needs to construct word vector model, while character-based model does not. We can also find that machine learning model cost more time in test stage compared with time costed in train stage, the main reason is that deep learning model train multi-epochs in training stage.

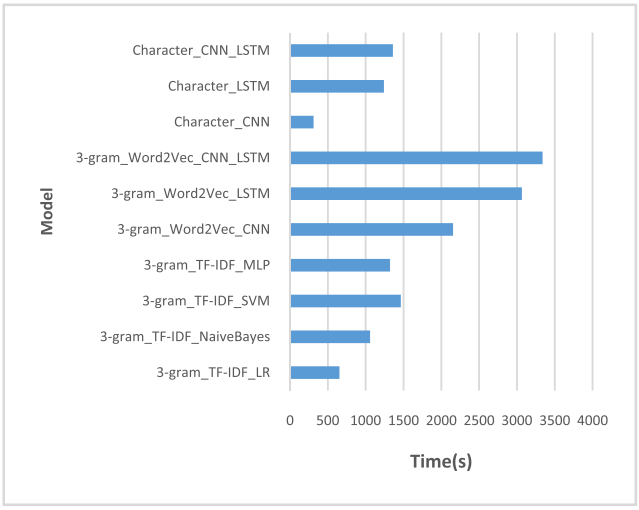

FIGURE 8. Training time of different models.

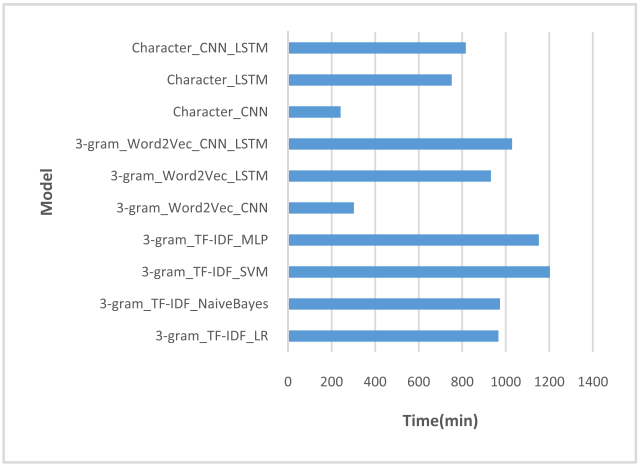

FIGURE 9. Test time of different models.

F. GENERALIZATION ABILITY VERIFICATION

In order to verify the generalization ability of the model in mass traffic detection in real environment, we have done some experiments. Considering the fact that the actual traffic volume is large and without labels, so it is difficult to test its generalization ability. In our experiments, we use the rule based baseline result to assist manual verification.

TABLE 5. Units for magnetic properties.

Model	*OWT*	*CC*	*ESC*	*SSF*	Out of Rule	Malicious Percent
Rule Baseline	16344	1522	2492	68698		
3-gram_TF-IDF_LR	12879	1109	1455	32677	18933	47%
3-gram_TF-IDF_NaiveBayes	13564	1325	1699	34430	65338	15%
3-gram_TF-IDF_SVM	13142	1265	1543	35231	26544	43%
3-gram_TF-IDF_MLP	11834	1123	1583	37692	19602	54%
3-gram_Word2Vec_CNN	14378	1445	1792	47932	37356	46%
3-gram_Word2Vec_LSTM	14209	1489	1821	48055	33342	50%
3-gram_Word2Vec_CNN_LSTM	14758	1492	1845	48997	31339	54%
Character_CNN	15528	1447	1889	50935	39042	65%
Character_LSTM	15402	1442	1905	49802	31635	70%
Character_CNN_LSTM	15923	1479	1987	54294	38289	75%

We extract an unlabeled dataset with 10 million traffic entries from http traffic monitor. Firstly, we feed these data into Webshell rule based detection system to generate a baseline result. The Webshell types generated from rule based detection system include *One Word Trojan(OWT), China Chopper(CC), Execute Shell Command(ESC) and Scan Shell File(SSF)*. The result is shown in Table 5.

We find that the character level CNN combined with LSTM model maintain a good generalization ability in actual traffic. We manually checked the malicious traffic detected out by each models, which can evade the detection engine based on rules. It should be pointed out that malicious percent represents the proportion of all data outside the rules that are manually verified as Webshell requests to all out-of-rule data. For example, the character-based CNN_LSTM model detect 38,289 out-of-rule abnormal traffic, and manual tests show that 75% of those are Webshell traffic entries. For comparison, using word2vec based CNN_LSTM model, detection number is 31,339, of which only 54% are Webshell requests, the other are model misjudgment requests, which show that character-based model is superior to word2vec based model in recall rate and precision rate. So we can find that the model we proposed is available in real environment. At the same time, it contains the ability to discover unknown Webshell threats.

VI. RELATED WORK

Many works have been done in Webshell detection and diagnose. The research work can be divided into two directions, one is based on the host, and the other is based on network traffic.

Host based detection usually uses eigenvalues and hazard functions to detect Webshell files [16], [25]–[28]. If a reasonable rule is found, this method could achieve high success rate of detection with easy operations, and quickly detect the presence of Webshell. Kong *et al.* [26] proposed the Webshell detection method based on *SimHash* algorithm [29]. They build up a *SimHash* fingerprint library based on Webshell code reuse, which combining with the idea of Webshell code reuse in building up *SimHash* fingerprint library, can be applied to small and medium size website for real-time detection and alarm. The PHP opcode(operation code) sequences are important features applied for PHP based Webshell detection [27]. A PHP Webshell detection model based on a combination of fastText [30] and random forest algorithm [31] is proposed to learn the opcode sequences features [27]. Ying and Yong [19] proposed a Webshell detection method based on correlation analysis according to the obfuscation statistical characteristics of Webshell.

However, such matching method only achieves high success rate with some existing Webshell, for some of the latest Webshell, the recall rate will be relatively low, and it almost couldn't detect 0day Webshell. What's more, because this method is specific to the hosts, it cannot meet the detection requirements of large-scale Webshell events.

Network based detection could monitor the request and response traffic generated in the communication activities, system commands, and status changes, to find abnormal behaviors and detect the existence of Webshell [2], [17], [32], [33]. Ye *et al.* [2] analyze the structural and textual features of pages, use bag-of-words model to extract keywords, and then use the SVM method to classify and detect Webshell. Sun *et al.* [33] analyze the different features of a page and propose a novel matrix decomposition based Webshell detection algorithm which can make predictions on the unknown pages. Tian *et al.* [17] propose a new malicious Webshell detection approach based on word2vec representation and convolutional neural network (CNN). The effectiveness of model checking and practical application can be further optimized.

VII. CONCLUSIONS

In this paper, we analyze Webshell attacks from traffic and try to find unknown Webshell attacks. We propose a character-based feature extraction method for sequential content. Based on this method, we obtain feature vectors as the input of the combined model of CNN and LSTM proposed in this paper,

we construct testbed in a variety of experimental scenarios and verify that the feature extraction and model in this paper have a good performance in precision, recall, F1-score, AUC, and generalization ability. This paper does not analyze the Webshell behavior pattern, this is our next step.

REFERENCES

[1] (2017). *China Internet Network Security Report*. [Online]. Available: http://www.cert.org.cn/publish/main/46/2018/20180802135136854322283/20180802135136854322283_.html
[2] F. Ye, J. Gong, and W. Yang, "Black box detection of webshell based on support vector machine," *J. Nanjing Univ. Aeronaut. Astronaut.*, vol. 47, no. 6, pp. 924–930, 2015.
[3] B. Scott and H. Ben. *Web Shell Detection Using NeoPI*. [Online]. Available: http://resources.infosecinstitute.com/web-shell-detection/
[4] Y. Chen, H. Jiang, C. Li, X. Jia, and P. Ghamisi, "Deep feature extraction and classification of hyperspectral images based on convolutional neural networks," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 10, pp. 6232–6251, Oct. 2016.
[5] S. C. Dos and M. Gatti, "Deep convolutional neural networks for sentiment analysis of short texts," in *Proc. 25th Int. Conf. Comput. Linguistics, Tech. Papers.*, 2014, pp. 69–78.
[6] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," in *Proc. INTERSPEECH*, 2012, pp. 601–608.
[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[8] *Kali Official Website*. [Online]. Available: https://www.kali.org/
[9] R. Rivest, *The MD5 Message-Digest Algorithm*, document RFC 1321, 1992, vol. 473, no. 10, p. 492.
[10] D. Eastlake and P. Jones, *US Secure Hash Algorithm 1 (SHA1)*, document RFC 3174, 2001.
[11] S. Banerjee and T. Pedersen, "The design, implementation, and use of the Ngram statistics package," in *Proc. 4th Int. Conf. Comput. Linguistics Intell. Text Process.*, vol. 2588, 2003, pp. 370–381.
[12] G. Salton and C. T. Yu, "On the construction of effective vocabularies for information retrieval," in *Proc. Meeting Program. Lang. Inf. Retr.*, New York, NY, USA, 1973, pp. 48–60.
[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *Comput. Sci.*, Sep. 2013.
[14] T. Joachims, "A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization," in *Proc. Int. Conf. Mach. Learn.*, 1996, pp. 143–151.
[15] R. C. Chen and S. P. Chen, "Intrusion detection using a hybrid support vector machine based on entropy and TF-IDF," *Int. J. Innov. Comput., Inf., Control*, vol. 4, no. 2, pp. 413–424, 2008.
[16] J. Lilleberg, Y. Zhu, and Y. Zhang, "Support vector machines and word2vec for text classification with semantic features," in *Proc. IEEE 14th Int. Conf. Cognit. Inform. Cognit. Comput.*, Jul. 2015, pp. 136–140.
[17] Y. Tian, J. Wang, Z. Zhou, and S. Zhou, "CNN-Webshell: Malicious Web shell detection with convolutional neural network," in *Proc. 6th Int. Conf. Netw., Commun. Comput. ACM*, 2017, pp. 75–79.
[18] X. Rong, "Word2vec parameter learning explained," *Comput. Sci.*, Nov. 2014.
[19] Z. Ying and H. Yong, "Webshell detection method based on correlation analysis," *J. Inf. Secur. Res.*, vol. 4, no. 3, pp. 251–255, 2018.
[20] V. Laippala and F. Ginter, "Syntactic n-gram collection from a large-scale corpus of Internet finnish," in *Proc. 6th Int. Conf. Human Lang. Technol. Baltic Perspective Baltic HLT*. IOS Press, vol. 268, 2014, p. 184.
[21] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 649–657.
[22] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 807–814.
[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn. (JMLR)*, 2015, pp. 448–456.
[24] G. Hinton, N. Srivastava, A. Krizhevsky, R. R. Salakhutdinov, and I. Sutskever, "Improving neural networks by preventing co-adaptation of feature detectors," *Comput. Sci.*, vol. 3, no. 4, pp. 212–223, Jul. 2012.
[25] H. Z. Du and Y. Fang, "PHP webshell real-time dynamic detection," *Netw. Secur. Technol. Appl.*, 2014.
[26] D. G. Kong *et al.*, "Detection method of webshell based on Simhash algorithm," *Commun. Technol.*, vol. 3, 2018.
[27] Y. Fang, Y. Qiu, L. Liu, and C. Huang, "Detecting webshell based on random forest with fasttext," in *Proc. Int. Conf. Comput. Artif. Intell.*, 2018, pp. 52–56.
[28] Z. Meng, R. Mei, T. Zhang, and W.-P. Wen, "Research of Linux webshell detection based on SVM classifier," *Netinfo Secur.* vol. 5, p. 004, 2014.
[29] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proc. 34th ACM Symp. Theory Comput. ACM*, 2002, pp. 380–388.
[30] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. (Aug. 2016). "Bag of tricks for efficient text classification." [Online]. Available: https://arxiv.org/abs/1607.01759
[31] L. Breiman, "Random forest," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.
[32] H. U. Jiankang, X. U. Zhen, M. A. Duohe, and Y. Jing, "Research of webshell detection based on decision tree," *J. Netw. New Media*, vol. 6, p. 005, 2012.
[33] X. Sun, X. Lu, and H. Dai, "A matrix decomposition based webshell detection method," in *Proc. Int. Conf. Cryptogr., Secur. Privacy. ACM*, 2017, pp. 66–70.

HUA ZHANG received the B.S. degree in communication engineering from Xidian University in 2002, the M.S. degree in cryptology from Xidian University in 2005, and the Ph.D. degree in cryptology from the Beijing University of Posts and Telecommunications (BUPT) in 2008. She is currently an Associate Professor with the Institute of Network Technology, BUPT. Her research interests include cryptography and information security.

HONGCHAO GUAN received the B.S. degree in network engineering from the Beijing University of Posts and Telecommunications in 2016, where he is currently pursuing the M.S. degree in computer science and technology. His research interests include network security, natural language processing, deep learning, and big data mining.

HANBING YAN received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 2006. He is currently with the National Computer Network Emergency Response Technical Team/Coordination Center of China. His research interests include cybersecurity, image analysis, and computer graphics.

WENMIN LI received the B.S. and M.S. degrees in mathematics and applied mathematics from Shaanxi Normal University, Xi'an, Shaanxi, China, in 2004 and 2007, respectively, and the Ph.D. degree in cryptology from the Beijing University of Posts and Telecommunications, Beijing, China, in 2012. She is currently a Lecturer with the Beijing University of Posts and Telecommunications. Her research interests include cryptography and information security.

YUQI YU received the B.S. degree in information management and information system from the China University of Petroleum, Beijing, in 2016. She is currently pursuing the M.S. degree in management science and engineering with Beihang University. Her research interests include machine learning, data mining, and Web security.

HAO ZHOU received the M.S. degree in information and communications engineering from the Beijing University of Post and Telecommunications. He is currently an Engineer with the National Computer Network Emergency Response Technical Team. His research interests include machine learning, cybersecurity, and image processing.

XINGYU ZENG received the B.S. degree in network engineering from Beijing Information Science and Technology University in 2015. He is currently pursuing the Ph.D. degree with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, China. His current research interests focus on network security, machine learning, deep learning, and big data mining.