Hindawi
Security and Communication Networks
Volume 2019, Article ID 3093809, 11 pages
https://doi.org/10.1155/2019/3093809

Research Article

Session-Based Webshell Detection Using Machine Learning in Web Logs

Yixin Wu,[1] Yuqiang Sun,[1] Cheng Huang,[1] Peng Jia,[2] and Luping Liu[2]

[1]*College of Cybersecurity, Sichuan University, Chengdu, China*
[2]*College of Electronics and Information Engineering, Sichuan University, Chengdu, China*

Correspondence should be addressed to Cheng Huang; opcodesec@gmail.com

Received 2 April 2019; Revised 14 July 2019; Accepted 24 July 2019; Published 22 November 2019

Guest Editor: Sebastian Schrittwieser

Attackers upload webshell into a web server to achieve the purpose of stealing data, launching a DDoS attack, modifying files with malicious intentions, etc. Once these objects are accomplished, it will bring huge losses to website managers. With the gradual development of encryption and confusion technology, the most common detection approach using taint analysis and feature matching might become less useful. Instead of applying source file codes, POST contents, or all received traffic, this paper demonstrated an intelligent and efficient framework that employs precise sessions derived from the web logs to detect webshell communication. Features were extracted from the raw sequence data in web logs while a statistical method based on time interval was proposed to identify sessions specifically. Besides, the paper leveraged long short-term memory and hidden Markov model to constitute the framework, respectively. Finally, the framework was evaluated with real data. The experiment shows that the LSTM-based model can achieve a higher accuracy rate of 95.97% with a recall rate of 96.15%, which has a much better performance than the HMM-based model. Moreover, the experiment demonstrated the high efficiency of the proposed approach in terms of the quick detection without source code, especially when it only considers detecting for a period of time, as it takes 98.5% less time than the cited related approach to get the result. As long as the webshell behavior is detected, we can pinpoint the anomaly session and utilize the statistical method to find the webshell file accurately.

1. Introduction

Webshells have become the main threat challenges for protecting the security of websites. According to the weekly safety report issued by National Computer Network Emergency Response Technical Team/Coordination Center of China (CNCERT/CC) in 2019, the number of websites with backdoors is growing almost every week [1]. As a web service-based backdoor program, webshell is installed through vulnerabilities in web applications or weak server security configuration such as SQL injection, the file including and uploading. Webshells with encryption and obfuscation are often used in attacks mostly because they are difficult to detect by WAF and other antivirus software. A hacker can initiate attacks using webshell tools such as Chinese Chopper [2] and achieve quite a few large malicious attacks like data theft, DDoS attacks, and watering hole attacks [3].

When a malicious webshell attack occurs, there are some exceptions as alertness for admin such as files with an abnormal timestamp, high traffic for a user in very short period time, and files including malicious codes. Attackers can also hide webshell logins in fake error pages.

Due to the high usage of webshell in cyberattacks, there has been much previous research in this field. But most researchers focus on the contents of suspicious files [4–6] or POST contents in HTTP requests [7] and thus ignore features in a sequence of web server logs. With the development of encryption and obfuscation technology [8, 9], it is quite difficult to detect webshell in thousands of website source files, but when we only need to deal with the sequence of several fields in the web logs without considering complex text processing, the problem becomes much simpler. This paper focuses on providing a new webshell detection method based on user's sessions in web logs to improve the accuracy

of webshell detection simply and effectively. Moreover, if the website defenders can identify malicious webshell immediately, they can prevent related cyber threats.

This paper presents a comprehensive framework including log data collection, feature extraction, session identification, and comparison of two machine learning methods. The main innovations of this paper can be summarized as follows:

(i) The presented model utilized a session-based method to detect webshell which is simpler and more efficient than existing methods. Session identification can be more precise using a statistical method which roughly calculates the time interval of each entry in each session at first and then set the threshold by the quantile in the statistics to divide the session more detailed. The experiment indicates that the model obtained the highest accuracy and recall rate when the threshold was 70% quantile.

(ii) The paper compared the long short-term memory with the hidden Markov model which are commonly used to process sequence data, and it suggested that the long short-term memory model which can provide a high accuracy rate of 95.97% with a recall rate of 96.15% has much better performance than the latter one.

(iii) The proposed model could only need access log files to check the malicious behavior of webshell without any web application source files. The results show that the LSTM-based model takes 98.5% less time than previous related approach to detect whether a website has a webshell attack in a period of time.

As a remainder, related work is briefly discussed in Section 2. A complete framework is explained in Section 3. Section 4 presents the entire experiment and evaluation process in detail. At last, the conclusion is in Section 5.

2. Related Work

2.1. Outline of Webshell. Webshell is a kind of software which usually assists the administrator to manipulate the server. But in some cases, attackers will use some malicious webshells to control the server to achieve malicious purposes. For attackers, webshell is a kind of backdoor, generally written in some scripting language like ASP, PHP, or JSP. These kinds of web scripts are able to create dynamic interactive sites, so the attacker is able to control the web server through web pages. These behaviors will be recorded in the access log [10].

2.1.1. Principle of Webshell. Webshell can work by sending HTTP requests to the specific page and use some functions to execute the instruction sent by the attacker. The results of these instructions will be sent to the attacker as the response of these HTTP requests [11].

2.1.2. Classification of Webshell. According to the function and size of scripting language, webshell can be roughly divided into three categories as follows:

(i) *Big Trojan.* It has a large size and comprehensive functions for command execution, database operations, and other malicious intentions. Besides, a friendly graphical interface is applied to the big Trojan.

(ii) *One Word Trojan*. It is a Trojan with only one line of code. Due to its shortness, it is often embedded in normal files or pictures. It can perform many functions like the big Trojan when connected to the initial attack tools such as Chinese Chopper.

(iii) *Small Trojan*. It is small and easy to hide but generally only has an upload function. Since most websites have size restrictions when uploading files, attackers generally first obtain upload permission through the small Trojan and then upload the big Trojan to the website to perform key functions.

2.1.3. Escape of Webshell. In order not to be discovered by the administrator of the website, webshells usually undergo a lot of distortion. Some escape methods are as follows:

(i) Pass parameters with less common fields: while websites generally use request field to pass parameters, this method uses some unusual fields such as HTTP referrer and user agent.

(ii) Encrypt sensitive features: attackers use some common encryption algorithms such as base64 [12] and rot 13 to encrypt some key functions. For a greater probability of escape, some tools even customize encryption algorithms.

(iii) Multiple encoding and compression: hackers change the original static features of the code by multiple encoding combined with compression technology to reduce the possibility of detection.

2.2. Method of Webshell Detection. We did some research on the previous detection of webshell. At the earliest, webshell detection takes a manual identification method. This is the oldest and most traditional way to detect webshells, which places high demands on the administrators of the website. Administrators are supposed to have a comprehensive grasp of the website files and have a high recognition ability for some newly added exception files [13], such as some naming files, passby.php, pass.asp, and a.jsp. Besides, these small files should be treated carefully because there are probably one word Trojans. After finding suspicious files, we need to analyze the contents of the file. The most thorough way is to take a look at the entire file carefully, but it will take a bunch of time. A better way is to search for some sensitive functions such as *exec*(), *shell_exec*(), and *system*() and check their parameters carefully [14].

Static feature webshell detection is the hot trend of research. It is an upgraded version of manual identification, but they are almost identical. This method focuses on the features of file contents. Due to numerous features, it often uses machine learning to improve effectiveness and accuracy. For example, in [15], an approach based on optimal

thresholds was proposed to identify files containing malicious codes from web applications. The detection system will scan and find malicious codes in each file of the web application, analyzing the features including keywords, file permissions, and owner. Instead of the source file codes, Tian et al. [16] divided the POST contents in the HTTP request into several words, which were represented in the form of vectors using the Word2vec model [17] and then were input into the CNN in a fixed-size matrix. Experiments have shown that such a method can achieve high accuracy, which was also the first time that CNN [18] applied to webshell detection. However, if we just detect by signature matching, we can only do well in detecting webshells that have known features, and it does not apply to detect unknown webshells. In the literature [19], an approach that can be used to predict unknown webshells was proposed. Supervised machine learning and matrix decomposition were used to generate original and unknown webshell features by analyzing different features of known pages. The paper [20] focuses on the detection of PHP webshell, which used the text classifier fastText [21] developed by Facebook and the random forest algorithm to build the model. As a compiled intermediate language of PHP scripts, PHP opcode sequence was regarded as an important feature of webshell detection. This paper can be a good inspiration because it starts to break away from the webshell file itself.

There are also some detection methods based on other features. For example, dynamic feature detection uses the system commands, network traffic, and state exceptions used by webshell to determine the threat level of the action. Webshell is usually confused and encrypted to avoid detection of static features. When the webshell is running, system commands must be sent to the system to reach the purpose of operating the database or even the system. This method monitors and even intercepts system commands by detecting system calls and deeply detects the security of the script from the behavior mode. Zhang et al. [22] proposed a character-level content feature transformation method, which combined the features of CNN and LSTM to construct a new webshell traffic detection model. This model preserves the sequential features in network traffic and reduces the feature dimension. Experiments have shown that this model can be used to detect unknown webshells while running on large websites. Besides, Shi et al. [23] proposed a log-based method for webshell detection, which uses features such as text features, statistical features, and page association about the degree of graph theory. But the session simply uses the IP field and the user-agent field for a rough distinction, which treats all access by a user as a session. The request field is used for machine learning, and it is a text processing problem like the static detection we mentioned earlier in essence.

Among the methods we mentioned above, the defenders need to scan and look for malicious codes inside every file of the web application or every POST content in the web logs [24]. The drawbacks of these methods are the heavy workload. What is worse, with the development of encryption and obfuscation techniques, webshell detection accuracy will be much lower because webshell detection tools are mostly based on signature matching. As we mentioned before, webshell mainly works by sending HTTP requests. Based on this theory, a new direction for webshell detection is proposed which focuses on using the raw sequence data without POST contents in web logs.

3. Framework

3.1. Architecture. The purpose of this paper is to detect webshell according to the sessions extracted from web server logs without POST contents or source file codes. It is composed of several components, as illustrated in Figure 1. Firstly, we use normal log files on a large website and collect honeypot logs with webshell communication for experimental data collection. In the second part, instead of extracting the fields directly from the web logs, we use the IP field and the user-agent field to roughly divide the collected logs into different sessions, count the time interval in every session, and set the threshold to identify the session in more detail first. Then, we use the hidden Markov model and long short-term memory to build our model, respectively, and compare which one has better performance.

3.2. Raw Data Collection. In the process of generating data, we use different kinds of webshells including big Trojans, small Trojans, and one word Trojans, which differ in size and function. Besides, these webshells are placed in different depths of the website we built, and different webshell tools such as Chinese Chopper and WeBaCoo are used to initialize the attack. In this paper, the log format of the website we use is Apache, but our framework can be applied to other common server log formats such as Nginx.

3.3. Data Processing

3.3.1. Data Extraction. All the logs we collected are in the common Apache2 format. There are close to 10 fields in this format, but there is no need to use all of them. Therefore, some fields were extracted from the web logs. The whole process is shown in Figure 2. Feature sequence consists of source IP address, timestamps, user agent, status code, bytes, referrer, and request, in which the request field is subdivided into method field and path field.

The IP field and user-agent field are used to identify sessions. Because there may be multiple visitors under the same IP, it is necessary to combine user agent to make judgments. At the same time, we roughly regard a visitor as a session for the time being. The status code field and byte field can be utilized to construct feature vectors without any changes. In the method field, the GET method is represented by 1, POST method is represented by 2, and other methods are represented by 0. Meanwhile, in the referrer field, "–" is represented by 0, while 1 for the website, 2 for the web crawler, and 3 for the external websites. In the timestamps field, we calculate the time difference between entries in every session and fill 0 in the last vector of every session.

The path field is encoded with the degree of relevance of each entry access path in the session. The first entry of each session is coded as −1 because there is no access in front of it,

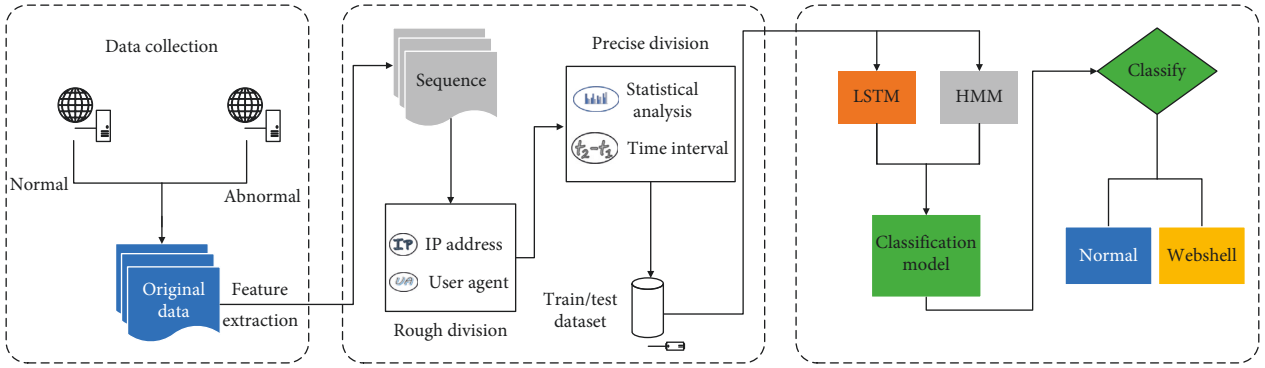

Figure 1: The architecture of the proposed method.

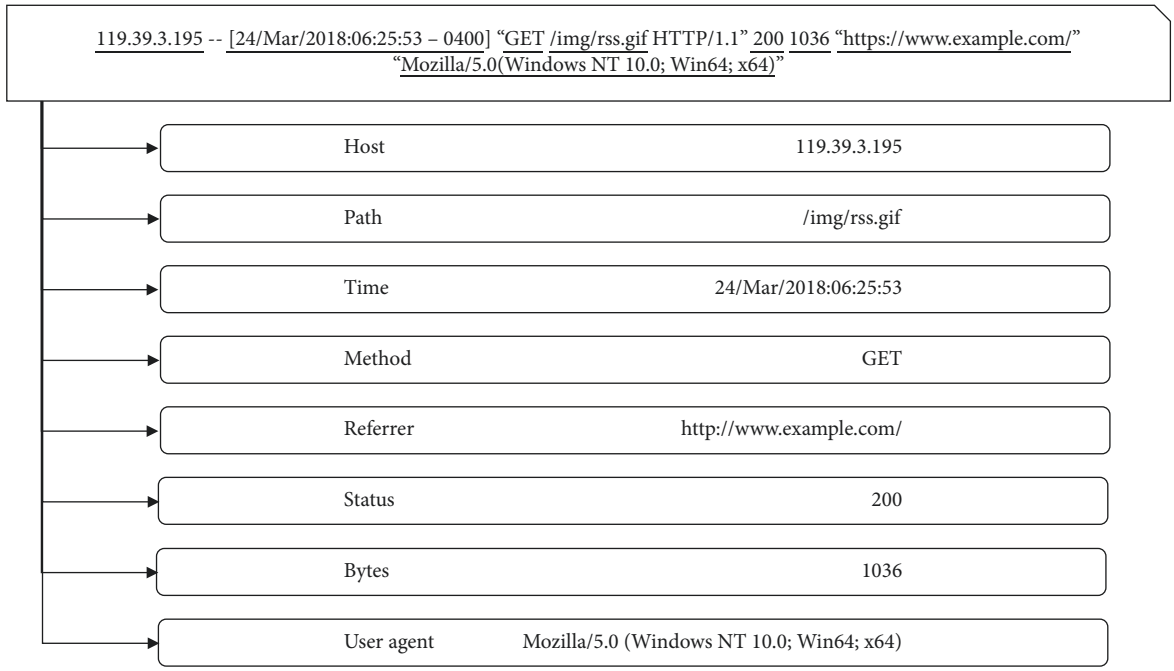

Figure 2: Feature extraction based on log entry.

and the relative relationship cannot be discriminated; from the second file, the access to the same file is marked as 0, and when accessing different files, the distance between different files is calculated by the number of directory switches plus one to encode. The process of encoding all paths for a session is shown in Figure 3. When all the feature fields are processed, a session will transform into a sequence of features as illustrated in Table 1. Finally, the feature vector we get will be six-dimensional including byte field, method field, path field, referrer field, status code field, and time interval.

3.3.2. Session Identification. In the previous section, we just made a rough distinction between the sessions in every log file, treating a visitor as a session. But more often, a visitor can access at different times and generate multiple sessions. Thus, we use a more scientific statistical method for session identification.

We calculate and count the time interval between entries in every session, which is roughly generated in the previous section. After counting the time intervals in all sessions, we sort all of them and explore the most appropriate quantile as a threshold. We compare every time interval to a threshold, and if the time interval is greater than the threshold, the session will be subdivided into two smaller sessions, and so forth. The whole process is illustrated in Figure 4.

When all the data have been processed, the framework will check all the sessions and delete those sessions that contain only one entry, as it assumes that if a user has only one access, it is highly unlikely that there is a webshell communication.

3.4. Classification Model. Webshell detection is a two-class task, and because of variable length serialization in the log sequence, we choose long short-term memory and the hidden Markov model to construct the classification model, respectively.

3.4.1. Long Short-Term Memory. Long short-term memory networks are well suited to classify and process based on

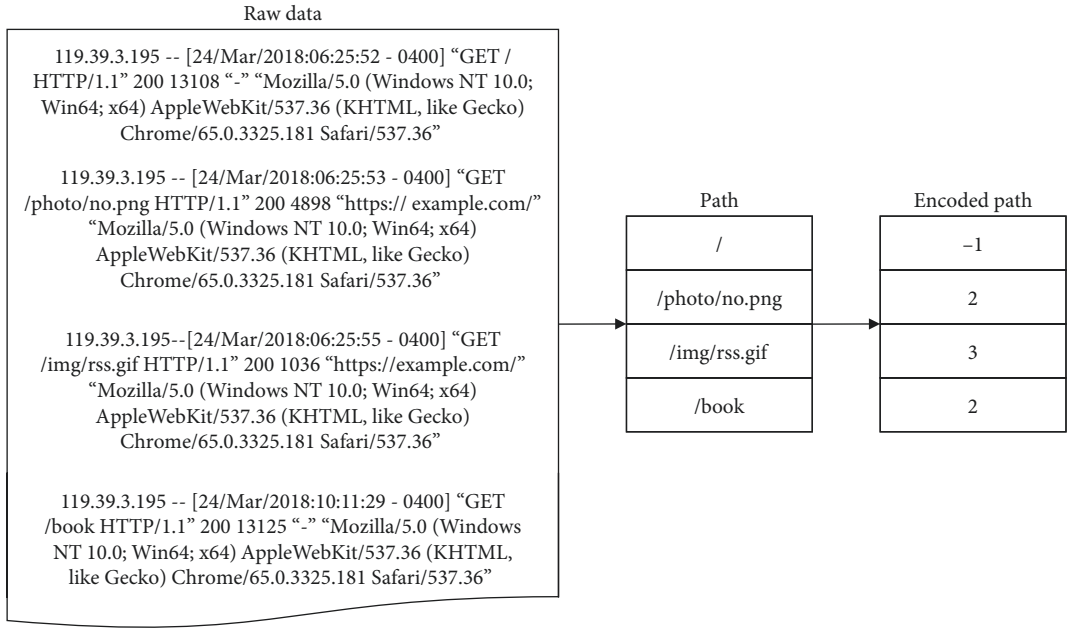

Figure 3: The process of encoding all paths for a session.

Table 1: Session transformation.

Log messages	Feature vector
Log entry0	$[bytes_0, method_0, path_0, referrer_0, statuscode_0, t_1 - t_0]$
Log entry1	$[bytes_1, method_1, path_1, referrer_1, statuscode_1, t_2 - t_1]$
Log entry2	$[bytes_2, method_2, path_2, referrer_2, statuscode_2, t_3 - t_2]$
⋮	⋮
Log entry$n$	$[bytes_n, method_n, path_n, referrer_n, statuscode_n, 0]$

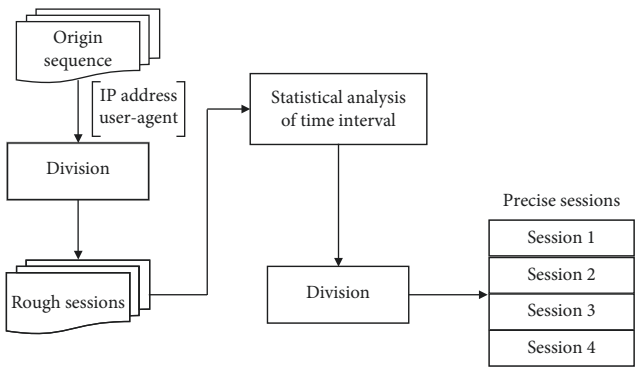

Figure 4: Schematic diagram of session identification.

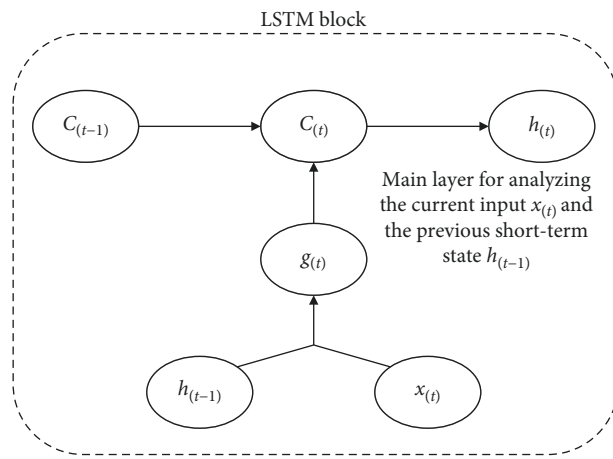

Figure 5: The core of LSTM.

sequence data with its faster convergence and the ability to detect long-term dependencies in data. The concept was originally proposed in 1997 by Hochreiter and Schmidhuber [25]. The emergence of LSTM is mainly to solve the problem of gradient disappearance and gradient explosion in long sequence training. Compared with traditional RNN where there is only one delivery state, the LSTM has two delivery states, namely, the long-term state $c_{(t)}$ and the short-term state $h_{(t)}$. Besides, it introduces three "gates" to control the long-term state, as illustrated in Figure 5.

(i) Forget gate $f_{(t)}$: control which of the long-term states of last moment $c_{(t-1)}$ should be discarded

(ii) Input gate $i_{(t)}$: control which parts of the current time input $x_{(t)}$ and the last short-term state $h_{(t-1)}$ will be added to the long-term state

(iii) Output gate $o_{(t)}$: control which parts of the current long-term state $c_{(t)}$ should be output as $h_{(t)}$

The model which called SB-LSTM (session-based long short-term memory) consists of four LSTM layers, each of which has sixteen neurons, and a dense layer. The LSTM layer will learn the feature of variable length sequences which are activated by tanh function and logistic function. The dense layer acts as a "classifier" throughout the LSTM, which can map the learned "distributed feature representation" to the sample tag space. At last, the detection model used categorical cross entropy as the loss function, and the optimizer is Adam.

In more detail, the input data size is $m \times n \times 6$, m is the number of sessions, and n is the maximum length of all sessions. Current long-term state $c_{(t)}$ and output $y_{(t)}$ are computed as follows:

$$\begin{aligned} g_{(t)} &= \tanh\left(W_{xg}^{T} \cdot x_{(t)} + W_{hg}^{T} \cdot h_{(t-1)} + b_g\right), \\ c_{(t)} &= f_{(t)} \otimes c_{(t-1)} + i_{(t)} \otimes g_{(t)}, \\ y_{(t)} &= h_{(t)} = o_{(t)} \otimes \tanh\left(c_{(t)}\right), \end{aligned} \tag{1}$$

where W_{xg} denotes the weight matrix of the main layer connected to the input vector $x_{(t)}$ of size 1×6, while W_{hg} denotes the weight matrix of the main layer connected to the previous short-term state $h_{(t-1)}$. b_g represents the coefficient of variation for the main layer.

The architecture of this model for a session is depicted in Figure 6.

3.4.2. Hidden Markov Model. The hidden Markov model (HMM) is a statistical Markov model, a highly effective means of modeling a family of unaligned sequences [26, 27], which describes a hidden Markov chain stochastically generating unobservable state sequences and then generating a stochastical observation sequence from each state. As its most remarkable features, the state at any time t depends only on the state of the previous moment, while it is independent of the time t, the state, and the observation at other times. Besides, it also assumes that observations at any time depend only on the state at that moment, independent of other observations and states.

In this paper, we mainly focus on supervised learning method of the hidden Markov chain. The train data include observation sequences O and corresponding state sequences I, which can be written as

$$\{(O_1, I_1), (O_2, I_2), \ldots, (O_s, I_s)\}. \tag{2}$$

We can use the maximum likelihood estimation method to estimate the parameters of the hidden Markov model:

(1) We assume that the sample is in the state i at time t and the frequency of transition to state j at time $t + i$ is A_{ij}, and then the probability estimate of the transition state is computed as follows:

$$\widehat{a_{ij}} = \frac{A_{ij}}{\sum_{j=1}^{N} A_{ij}}, \quad i = 1, 2, \ldots, N;\ j = 1, 2, \ldots, N. \tag{3}$$

(2) We assume that the sample state is j and the frequency of observation k is B_{jk}, and then the probability of observation k when the state is j can be written as

$$\widehat{b_{jk}} = \frac{B_{jk}}{\sum_{k=1}^{M} B_{jk}}, \quad j = 1, 2, \ldots, N;\ k = 1, 2, \ldots, M. \tag{4}$$

(3) The initial state probability π is estimated as the frequency of the initial state of i in the S samples. The architecture of this model is illustrated in Figure 7.

In more detail, the input data size is $x \times 6$ and x is the number of entries in all sessions.

4. Experiment and Evaluation

In this section, we will introduce the detailed composition of the dataset, the experiment details, and the results.

4.1. Dataset. There are two datasets for our experiments, one for the model training and the other for the model test. As for the first dataset, a honeypot [28, 29] was built to capture and analyze webshell attacks. Tens of testers were invited to attack this honeypot. One word Trojans, small Trojans, and big Trojans which were coded in different program languages were provided to these testers. We collected a total of 13,986 log entries from the honeypot as negative samples, while 57,160 logs entries were collected from real-world websites as positive samples. In the process of training, we also used a 10-folder cross validation to perform a simple verification of the model. As for the latter one, a real-world website which contains a total of 2324 files and 248 k lines of log entries in total was utilized to test. Since the log entries with webshell communication in the real environment may only account for a few percents or even a few thousandths in the log file, we control the positive and negative sample ratio of the dataset to be around 6 : 1 in experiments. After feature vector extraction and session identification, all log entries become independent sessions. Every session consists of six sequences, which are the byte sequence of the HTTP response, the method sequence, the path sequence that is encoded by the degree of association, the referrer sequence that is divided into different cases for encoding, the status code sequence, and the time interval sequence. Figure 8 is an overview of our model.

4.2. Experiment Design. To evaluate the performance of our model, we first explored the effects of different quantiles as thresholds and selected the best values for next experiments. Then, we validated the effectiveness of the session identification method based on the time interval and quantile. In addition, we compare long short-term memory with the hidden Markov models, which are commonly used in sequence data processing, and find the one with better performance. Finally, we leverage a real-world website to test the SB-LSTM model and compare it with results from other research.

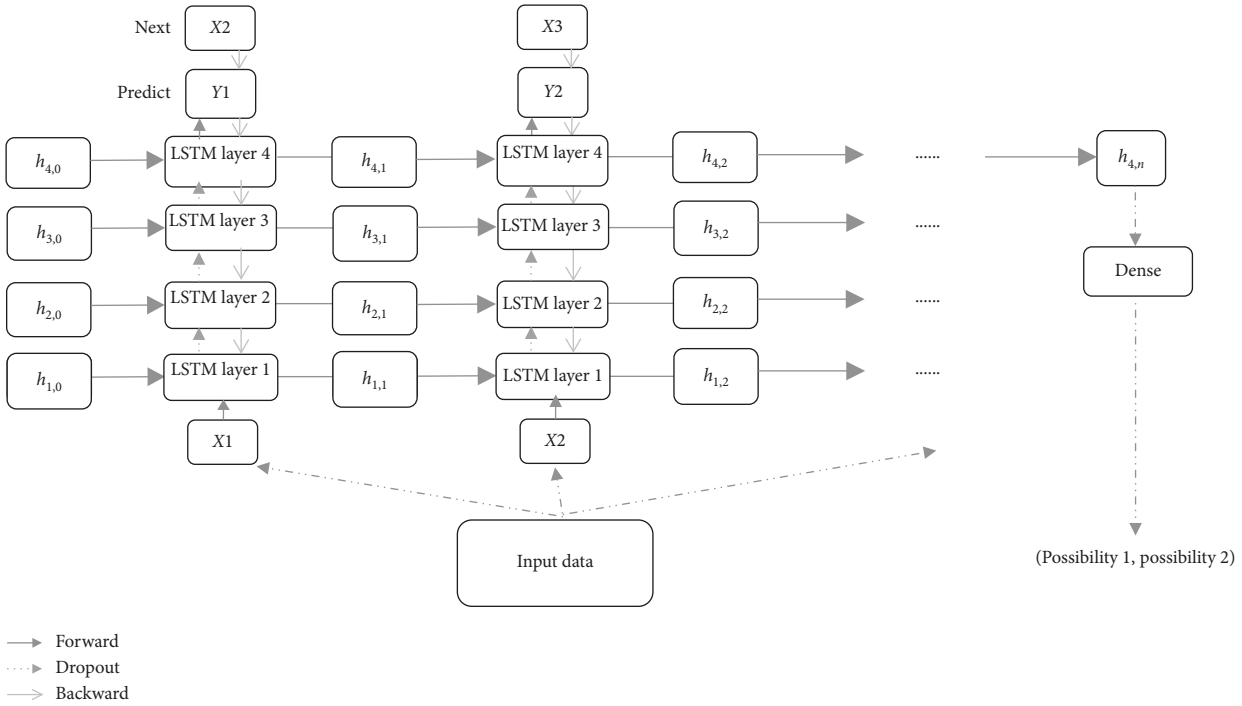

Figure 6: The architecture of LSTM.

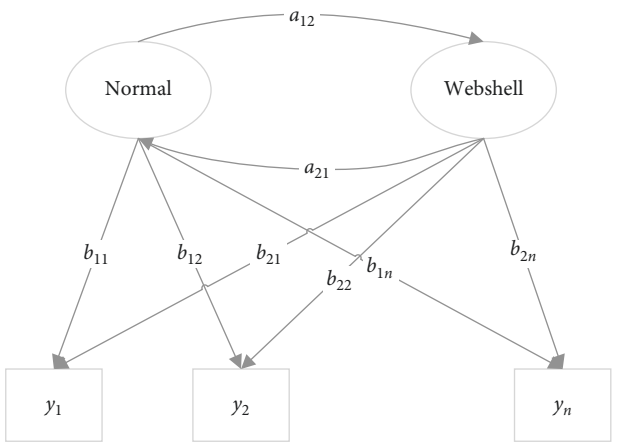

Figure 7: The architecture of HMM.

All the experiments were performed in a PC machine with an i7-7700HQ processor, 16 GB of memory, and a GeForce GTX 1060 GPU which has 6 GB of memory. The SB-LSTM model is implemented by Tensorflow [30], and the HHM is implemented by seqlearn [31].

Before using the raw data to build the model, we did a standardization to make all the data appropriate for the neural network. Data were processed according to the following steps:

(i) Since we need to input a 3-dimensional matrix when constructing the SB-LSTM model, we chose Max-Min scaling normalization to reduce the effect of padding 0 when normalizing. After scaling, all data are between 0 and 1. The function of doing Max-Min scaling normalization is

$$x^* = \frac{x - \min}{\max - \min}. \tag{5}$$

(ii) The label of all the training data is stored in a list, 0 for normal access and 1 for webshell access.

(iii) When we used long short-term memory to build our model, we must make all data to become a matrix, so we padded the length of the sequence to z, which is the maximum session length. Besides, we held a list to show all the valid length (original length) of the data to ensure the number of iterations in LSTM.

(iv) When we used the hidden Markov model to build our model, we only need to put all the vectors into a sequence because of the input data size of the hidden Markov model.

We use 10-folder cross validation [32]. The dataset is equally divided into 10 subsets, each of which is tested once and the rest as a training set. The cross validation is repeated 10 times, one subset is selected each time as a test set, and the average cross validation recognition accuracy rate of 10 times is taken as a result. The dataset used for the experiment is unbalanced, so we choose accuracy, recall, precision, $F1$ score [33], the receiver operating characteristics (ROC) curves [34], and area under curve (AUC) measure for evaluating the proposed method. Besides, we can visualize the relation between TPR and FPR of a classifier. These indicators can be expressed as

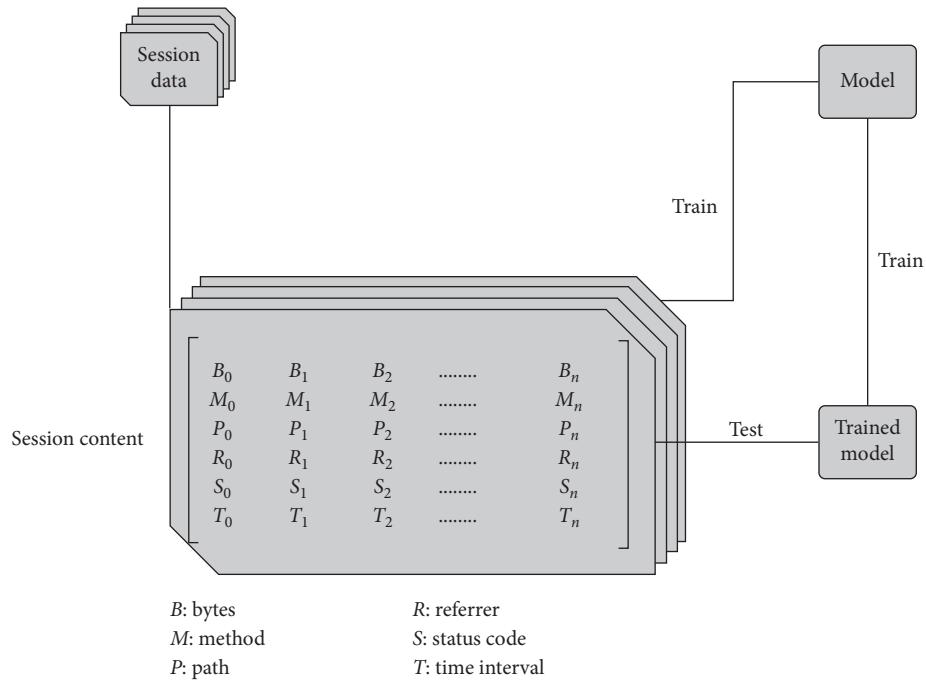

Figure 8: Overview of webshell detection with our model.

$$\begin{aligned}
\text{Accuracy} &= \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{\text{TP}_i + \text{TN}_i}{\text{TP}_i + \text{TN}_i + \text{FP}_i + \text{FN}_i},\\
\text{Recall} &= \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i},\\
\text{Precision} &= \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i},\\
F1\ \text{score} &= 2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}},\\
\text{AUC} &= \frac{\sum_{i\in\text{positiveclass}}\text{rank}_i - (M(1+M)/2)}{M \times N}.
\end{aligned} \tag{6}$$

where M is the number of positive samples and N is the number of negative samples. The score indicates the probability that each test sample belongs to a positive sample, while *rank* is a positive sample set sorted in the descending order based on score.

4.3. Experiment Results. First, we explored the influence of different thresholds in session identification. The thresholds were set to 55%, 60%, 65%, 70%, 75%, 80%, 85%, and 90% quantile, respectively. The comparison results are shown in Table 2.

As is shown in Table 2, performance is improved with an increasing threshold in a certain range and reaches the highest value when the threshold is 70%. Moreover, when the performance reaches the highest, it will continuously decrease with a rising threshold. Excessive thresholds can decrease the performance of the model because two accesses that were not originally part of the same session are placed in the same session. In comparison with the threshold of 55% to 90%, we find the appropriate threshold is 70%, and we use it for the next experiments.

Table 2: The influence of different thresholds in session identification.

Threshold (%)	Accuracy	Recall	Precision	$F1$ score
55	0.932	0.8624	0.8757	0.8690
60	0.9364	0.869	0.8958	0.8822
65	0.9598	0.9568	0.8977	0.9263
70	0.9597	0.9615	0.9036	0.9317
75	0.9696	0.9368	0.9009	0.9185
80	0.9376	0.9401	0.8518	0.8937
85	0.8974	0.7228	0.6932	0.7077
90	0.8779	0.7186	0.6418	0.6780

Then, we validated the effectiveness of session identification based on the time interval and the quantile. We counted the number of access sessions for every user, where the user is identified by the IP field and the user-agent field. The top 100 users with the largest number of sessions are shown in Figure 9.

As is illustrated in Figure 9, most users have multiple sessions, up to a maximum of 166. Therefore, it is quite necessary for us to conduct a more detailed session identification.

In addition, we compare the performance of two machine learning models. The comparisons are presented in Table 3.

As is shown in Table 3, SB-LSTM has a much better performance than the HHM-based model. The recall rate of the HMM-based model under such dataset training is close to 0. The reason that the HMM assumes that the state at any

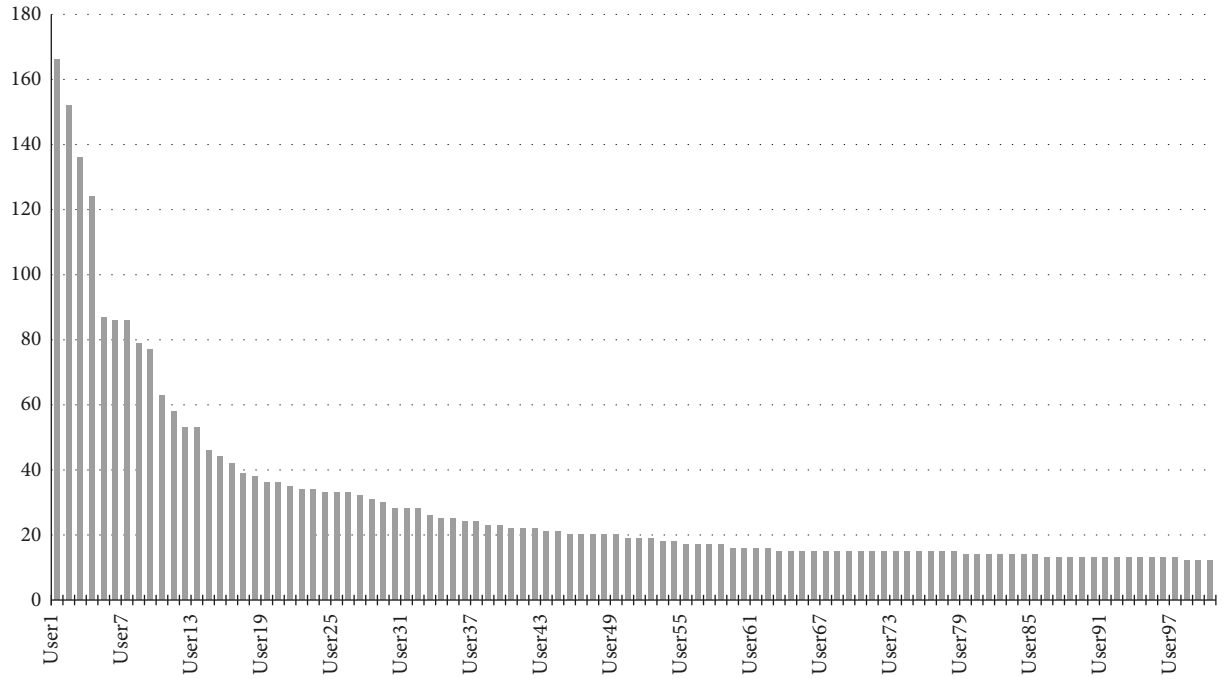

Figure 9: The top 100 users with the largest number of sessions.

Table 3: Comparison of two machine learning methods.

Model name	Accuracy (%)	Recall (%)
SB-LSTM	95.97	96.15
HMM-based model [31]	68.27	0.00

time depends only on the state of the previous moment might lead to the low recall. Besides, the HMM-based model also has a much lower accuracy than SB-LSTM. The ROC is shown in Figure 10. The curve shows the SB-LSTM model could achieve an effective and accurate result in which the true positive rate could reach 0.8 when the false positive rate only reaches 0.01.

Afterwards, we found the webshell communication, and we can easily find the webshell by counting all the access paths of the session. Besides, if the administrator is familiar with the files included in website dictionary, he may not need to perform statistics to find the webshell.

At last, we leverage a real-world website to test the SB-LSTM model and compare it with results from other research. We first explored the influence of the different sizes of logs on the model's runtime. The experiment used two log files including a recent one and the largest one the website can provide. Besides, we ran this experiment three times and got the average as results. The results are presented in Table 4. Then, we compared these to the result of the cited related work. We downloaded all the source files of the website and reproduced a cited related work, which is a file-based detection method. For the purpose of maximizing the experimental results, we used the largest log the website can provide which records all access to the website for nearly a year and a half to compare. We do the same things in terms of the final results, and the comparison results are shown in Table 5.

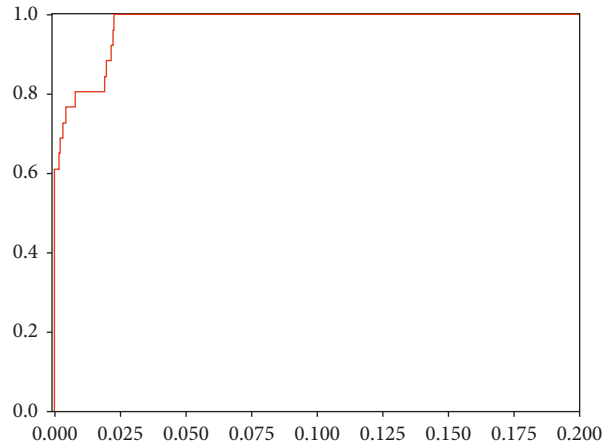

Figure 10: ROC curve based on SB-LSTM.

Table 4: The influence of the different sizes of logs.

Model name	Sizes	Entries	Time
SB-LSTM	177 kB	908	0.5897 s
	46.6 MB	248433	28.7127 s

Table 5: Comparison of different detection methods.

Model name	Sizes/amount	Time
SB-LSTM	46.6 MB	28.7127 s
FRF-WD [20]	2324	39.8415 s

As is illustrated in Table 4, when the size of the log is quite small, SB-LSTM can get the result of detection very quickly, which is efficient for detecting whether there is webshell communication for a certain period of time.

As revealed in Table 5, even if we use all the log entries since the website was built, the runtime required for analyzing is 27.9% less than the result of the cited related work.

Compared with the results of the first experiment, it demonstrated that the SB-LSTM model is efficient for detecting whether there is webshell communication for a certain period of time, as it requires to scan all the source files in the common approach while the only input for the SB-LSTM model is the recent log.

5. Conclusion

In the previous studies, a huge number of features were extracted from webshell, and these features were regarded as keywords to judge whether there is a webshell or not. However, it is almost certain that these approaches would be less useful with the gradual development of encryption and confusion technology. This paper mainly focuses on the challenge that detects webshell out of itself. Instead of leveraging POST contents, source file codes, or receiving traffic, the framework we proposed uses sessions generated from the website's logs, which highly reduces the cost of time and space but maintains a high recall rate and accuracy. Features were extracted in raw sequence data in the web logs, and a statistical method was applied to identify sessions precisely. The results of experiments show that 70% quantile can be the right threshold that makes the model obtain the highest accuracy and recall rate, and the long short-term memory which can achieve a high accuracy rate of 95.97% with a recall rate of 96.15% has much better performance than the hidden Markov model on webshell detection. Moreover, the experiment demonstrated the high efficiency of the proposed approach in terms of the runtime, as it takes 98.5% less time than the cited related approach to get the results. In order to be closer to the real-world application, the model can be employed to identify webshell files after the webshell communication is detected by using a statistical method.

Data Availability

The Web logs data used to support the findings of this study have not been made available because it was extracted from real websites, and contains many sensitive information.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the Fundamental Research Funds for the Central Universities and the Sichuan University Postdoc Research Foundation under Grant 19XJ0002.

References

[1] CNCERT weekly report," 2019, http://www.cert.org.cn/publish/english/upload/File/Weekly%20Report%20of%20CNCERT-Issue%207%202019.pdf.

[2] D. H. Tony Lee and I. Ahl, "Breaking down the China chopper web shell-part I," 2019, https://www.fireeye.com/blog/threat-research/2013/08/breaking-down-the-china-chopper-web-shell-part-i.html.

[3] B. Yong, X. Liu, Y. Liu, H. Yin, L. Huang, and Q. Zhou, "Web behavior detection based on deep neural network," in *Proceedings of the IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing*, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), pp. 1911–1916, Hong Kong, China, July 2018.

[4] L. Y. Deng, D. L. Lee, Y.-H. Chen, and L. X. Yann, "Lexical analysis for the webshell attacks," in *Proceedings of the 2016 International Symposium on Computer, Consumer and Control (IS3C)*, pp. 579–582, Xi'an, China, July 2016.

[5] V.-G. Le, H.-T. Nguyen, D.-N. Lu, and N.-H. Nguyen, "A solution for automatically malicious web shell and web application vulnerability detection," in *Proceedings of the International Conference on Computational Collective Intelligence*, pp. 367–378, Halkidiki, Greece, September 2016.

[6] J. Wang, Z. Zhou, and J. Chen, "Evaluating CNN and LSTM for web attack detection," in *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*, pp. 283–287, Macau, China, February 2018.

[7] W. Yang, B. Sun, and B. Cui, "A webshell detection technology based on http traffic analysis," *Innovative Mobile and Internet Services in Ubiquitous Computing*, in *Proceedings of the International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 336–342, Matsue, Japan, July 2018.

[8] J. Kim, D.-H. Yoo, H. Jang, and K. Jeong, "Webshark 1.0: a benchmark collection for malicious web shell detection," *JIPS*, vol. 11, no. 2, pp. 229–238, 2015.

[9] P. M. Wrench and B. V. Irwin, "Towards a php webshell taxonomy using deobfuscation-assisted similarity analysis," in *Proceedings of the 2015 Information Security for South Africa (ISSA)*, pp. 1–8, Johannesburg, South Africa, July 2015.

[10] Z. Meng, R. Mei, T. Zhang, and W.-P. Wen, "Research of linux webshell detection based on SVM classifier," *Netinfo Security*, vol. 5, pp. 5–9, 2014.

[11] O. Starov, J. Dahse, S. S. Ahmad, T. Holz, and N. Nikiforakis, "No honor among thieves: a large-scale analysis of malicious web shells," in *Proceedings of the 25th International Conference on World Wide Web*, pp. 1021–1032, Montreal, Canada, April 2016.

[12] S. Josefsson, "The base16, base32, and base64 data encodings," 2009, http://www.hjp.at/doc/rfc/rfc3548.html.

[13] Z.-H. Lv, H.-B. Yan, and R. Mei, "Automatic and accurate detection of webshell based on convolutional neural network," in *Proceedings of the China Cyber Security Annual Conference*, pp. 73–85, Beijing, China, August 2018.

[14] Compromised web servers and web shells-threat awareness and guidance," 2017, https://www.us-cert.gov/ncas/alerts/TA15-314A.

[15] T. D. Tu, C. Guang, G. Xiaojun, and P. Wubin, "Webshell detection techniques in web applications," in *Proceedings of the Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, pp. 1–7, Hefei, China, 2014.

[16] Y. Tian, J. Wang, Z. Zhou, and S. Zhou, "CNN-webshell: malicious web shell detection with convolutional neural network," in *Proceedings of the 2017 VI International*

Conference on Network, Communication and Computing, pp. 75–79, Kunming, China, December 2017.

[17] T. Walkowiak, S. Datko, and H. Maciejewski, "Bag-of-words, bag-of-topics and word-to-vec based subject classification of text documents in polish-a comparative study," *Contemporary Complex Systems and Their Dependability*, pp. 526–535, 2018.

[18] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[19] X. Sun, X. Lu, and H. Dai, "A matrix decomposition based webshell detection method," in *Proceedings of the 2017 International Conference on Cryptography, Security and Privacy*, pp. 66–70, Wuhan, China, March 2017.

[20] Y. Fang, Y. Qiu, L. Liu, and C. Huang, "Detecting webshell based on random forest with fasttext," in *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*, pp. 52–56, Chengdu, China, March 2018.

[21] Package Information [eb/ol], 2019, https://pecl.php.net/package/vld.

[22] H. Zhang, H. Guan, H. Yan et al., "Webshell traffic detection with character-level features based on deep learning," *IEEE Access*, vol. 6, pp. 75268–75277, 2018.

[23] L. Shi and Y. Fang, "Webshell detection method research based on web log," *Journal of Information Security Research*, vol. 1, p. 11, 2016.

[24] R. Sasi, "Web backdoors-attack, evasion and detection," in *Proceedings of the C0C0N Sec Conference*, pp. 989–1003, Cochin, India, 2011.

[25] J. Schmidhuber and S. Hochreiter, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[26] R. Hughey and A. Krogh, "Hidden markov models for sequence analysis: extension and analysis of the basic method," *Bioinformatics*, vol. 12, no. 2, pp. 95–107, 1996.

[27] B. Schuller, G. Rigoll, and M. Lang, "Hidden markov model-based speech emotion recognition," in *Proceedings of the 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Atlanta, GA, USA, April 2003.

[28] I. Kuwatly, M. Sraj, Z. Al Masri, and H. Artail, "A dynamic honeypot design for intrusion detection," in *Proceedings of the IEEE/ACS International Conference on Pervasive Services*, Lebanon, July 2004.

[29] C. Kreibich and J. Crowcroft, "Honeycomb," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 1, pp. 51–56, 2004.

[30] TensorFlow," 2019, https://www.tensorflow.org/.

[31] L. larsmans, "seqlearn," 2016, https://github.com/larsmans/seqlearn.

[32] R. Kohavi et al., "A study of cross-validation and bootstrap for accuracy estimation and model selection," *IJCAI*, vol. 14, no. 2, pp. 1137–1145, 1995.

[33] C. Goutte and E. Gaussier, "A probabilistic interpretation of precision, recall and *f*-score, with implication for evaluation, Lecture Notes in Computer Science," in *Proceedings of the European Conference on Information Retrieval*, pp. 345–359, Santiago de Compostela, Spain, March 2005.

[34] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006.