ATLANTIS PRESS

Advances in Computer Science Research, (ACSR), volume 52

2016 International Conference on Computer Engineering and Information Systems (CEIS-16)

The Research on BPNN-AdaBoost Model to Identify Web Servers

Zhen Huang, Chun-He Xia
Key Laboratory of Beijing Network Technology,
Beihang University,
Beijing, China
E-mail: hzhen@buaa.edu.cn, xch@buaa.edu.cn

Xiao Lu
National Computer Network Emergency Response,
Technical Team/Coordination Center of China,
Beijing, China
E-mail: luxiao@cert.org.cn

***Abstract*-In this paper, the BPNN-AdaBoost model is used for identification of Web servers to avoid the recognition of Web servers depending on banner information, reduce the cost of being maintained the feature library, and improve the accurate identification of Web servers. Through the model to identify Web servers, it not only can accurately identify the Web server in the case of hidden or forged Banner information, but also we need not maintain the feature library after the model is trained. Experimental results show the accurate identification of BPNN-AdaBoost model is much better than other methods, and the stability is better than the BPNN algorithm.**

Keywords-bpnn; adaboost; identify web servers

I. INTRODUCTION

The identification of Web servers has significant meaning for assessing security of Web system, researching market share and forecasting the scope of influence on a network security incident. The traditional way identifying Web servers through Banner information is simple and efficient. Now, there are a lot of softwares, such as Nessus and censys [1], they identify Web servers through this method. Generally, this method is considered to be effective. But with the increase of the security awareness of Web administrators, this method is faced with great challenges. Web administrators can alter the Web server's Banner by modifying the source codes and binary codes, or can modify and remove Banner information by using security plug-ins, such as ServerMask and mod_security. In this situation, identifying the Web server through the Banner information will be failed. Hmap [2] is based on the different features in lexical, grammatical, and semantic of HTTP response to identify the Web server. For example, in the lexical, the state code of 404 is described as "Not Found" by Apache, but Netscape-Enterprise described it as "Not found". However, OmniHTTpd called it as "not found Resource". In grammatical, for example, Microsoft-IIS/7.5 is returned to the server domain in front of the date domain, and the Apache/2.2.22 is opposed. But beyond that, according to Web servers solving long URL in different ways, it identified the target Web server through sending a large many long URL requests. Although the Web server can be identified by the differences in lexical and grammatical when target Web server hidden or forged Banner information, these information can also be modified by modifying the Banner way to judge it failed. Through sending HTTP requests for long URL to identify the Web server cannot be modified by simple text substitution, but the same type of Web servers tend to be handled in the similar way. So it is difficult to distinguish the Web server's version. Httprint is a Web server fingerprinting tool. It relies on Web servers' characteristics to accurately identify Web servers, despite the fact that they may have been obfuscated by changing the server banner strings, or by plug-ins. Httprint uses text signature strings to identify Web servers. However, identifying Web servers by strings of text signature, only all the features are matched, the string signature will be consistent. Some modification of the characteristic will be failure. Paper [3] proposes a method to judge the Web server according to different ways of processing protocol and version of the HTTP request without caring for other factors, such as special methods and special URLs. It is to identify the Web server by the matching of human maintained feature library, so it is difficult to distinguish the Web server when the similar number of features are matched. In this paper, we propose the BPNN-AdaBoost model to identify Web servers. By constructing a multi class of special HTTP requests, we can get Web servers' HTTP responses and extract Web servers' features. The model is trained by different features of Web servers in HTTP responses. After the model is trained, we can identify Web servers. The algorithm is proposed in this paper can not only effectively identify Web servers in the case of hidden or forged Banner information, we will also not need to maintain the feature library after the model is trained. More importantly, the accurate rate of identification is superior to other methods.

II. FEATURE CONSTRUCTION

A. Feature Extraction Basis

The interaction between the client and the Web server depends on HTTP. In order to maintain compatibility among Web servers, providers of the Web server must obey RFC standards. However, RFC [4,5] includes "MUST", "REQUIRED", "MAY", "SHOULD", "RECOMMENDED", "SHALL", and so on. An implementation that satisfies all the MUST level requirements not all the SHOULD level requirements for its protocols is said to be "conditionally compliant." When realizing the Web server, providers of the Web server generally follow the conditionally compliant. Therefore, some specific functions and implementation methods are different from other Web servers, even though all Web servers comply with RFC standards. According to the method of handling special HTTP requests and response information, we can identify the type and version of Web servers.

B. Feature Extraction Methods

In generally, different Web servers give a proper response for normal HTTP requests according to RFC standards. But when a client sends unreasonable HTTP requests to Web servers, different Web servers have different performances. This section describes how to extract features of the Web server through these special HTTP requests. The Request-Line starts with a method token, followed by a unified resource identifier and the protocol version, and ended with CRLF. The format is as follows.

Request-Line = Method SP Request-URL SP HTTP-Version CRLF

We introduce how to construct special HTTP requests to extract the characteristics of the Web server in four aspects, including request methods, Request-URL, protocol statement and protocol version.

1) Extracting features by methods: RFC defines eight methods kinds of request methods, including GET, HEAD, POST, PUT, OPTIONS, TRACE, CONNECT, and DELETE. RFC explicitly describes that the set of request methods supported by the Allow entity-header domain. The Allow header domain cannot prevent the client from using other methods. Methods from Allow header domain can be carried out, and different Web servers will be different in the implementation if the method does not in the Allow header field.Therefore, there are two methods to extract the characteristics of Web servers. One is defined method, but it is not in Allow header domain, the other is undefined method. We can extract the characteristic of Web servers according to the response when they deal with these situations. As we know, Web servers usually do not accept the DELETE request, but different Web servers have different handling modes. For example, the response to DELETE request is “405 Method Not Allow” for Microsoft-IIS/7.5, Netscape-Enterprise/6.0 thinks that the DELETE method is not authorized, and Apache-coyote/1.1 regards the DELETE method is forbidden. For undefined request methods, the processing mode of Microsoft-IIS/7.5 is the same as these methods which are not in the Allow domain. But the response of the Apache/2.2.22 and Netscape-Enterprise/6.0 for undefined request methods is “501 Method Not Implemented”. Therefore, we can extract characteristics of Web servers according to defined methods which are not in the Allow domain and methods undefined.

2) Extracting features by request-URL: The Request-URL is a Uniform Resource Identifier. Normally, the Web server responses the corresponding results according to Request-URL. But for some abnormal URLs, for example, the resource does not exist in Web servers or super long URLs, different Web servers have different ways to process these URLs. For example, the apache/2.0.26 return 403 for the URL does not exist but in a certain length, and return 414 when URL is more than 10000 characters. However, Microsoft-IIS/8.5 respective return 404 and 400. According to the solution of Web servers to deal with Request-URL, we can extract the characteristics of Web servers.

3) Extracting features by protocol statement: Although the RFC standards does not emphasize protocol declarations should be case-sensitive, and some Web servers think that the protocol which consists of lower-case letters is false. Due to "HTTP" is a default protocol, some Web servers ignore to examine the protocol in order to improve the processing speed. For example, Microsoft-IIS/7.5 only receives the capital HTTP. The Netscape-Enterprise/6.0 thinks protocol is case-insensitive. However, Apache/2.4.9 ignores to examine the protocol field. The characteristics of Web servers can be extracted according to different checking methods of the protocol statement in Web servers.

4) Extracting features by protocol version: HTTP uses a "<major>.<minor>" numbering scheme to indicate versions of the protocol. The major and minor numbers are regarded as separated integers. They stand for major version and minor version. According to RFC the major version should be denoted as 0 to 9. At present, the HTTP version include 0.9, 1 and 1.1. That is to say, the highest major version is 1. If the major version of the HTTP request is a number above 1, or not a number, different Web servers have different processing modes. Because there is no mandatory requirement in RFC for Web servers to handle these requests. For example, in the case that major version is above 1, the response of Netscape-Enterprise/6.0 is "505 HTTP Version Not Support ", Apache/1.3.27 is “400 Bad request”. In the case that major version is not a digit, the response of Netscape-Enterprise/6.0 is “Bad request”, but Apache/1.3.27 is “400 Bad request”. However, Microsoft-IIS/ 5.1 gives a normal response for these two cases. Similarly, different Web servers are not exactly the same in dealing with minor version, and we will not repeat them. Therefore, we can extract the characteristics of the Web server according to different situations of malformation protocol version.

In this paper, we construct 130 kinds of HTTP requests to extract features of Web servers through the above ways as well as their combination. And then we encode features as input to our model, and encode Web servers as the output of our model. Through training the model, the Web server can be identified.

III. MODEL CONSTRUCT

In recent years, neural networks have been widely used in many fields, and the achievements of neural networks in pattern recognition are particularly striking. Using self-learning, self-organization, fault tolerance and other characteristics of neural network, this paper adopts the error back propagation neural network (BPNN) model to identify the Web server. Because BPNN is a weak learning algorithm, there are the over fitting characteristic, the weak generalization ability, unstable and easily falling into local optima. On the contrary, Adaptive Boosting (AdaBoost) is a

kind of iterative algorithm, it can improve any given weak classifier accuracy. Aiming at the limitation of BPNN, and to improve the recognition accuracy. In this paper, we proposed a Web server identification algorithm based on BPNN-AdaBoost, which combines the AdaBoost algorithm with the BPNN. For a Web server sample and the corresponding

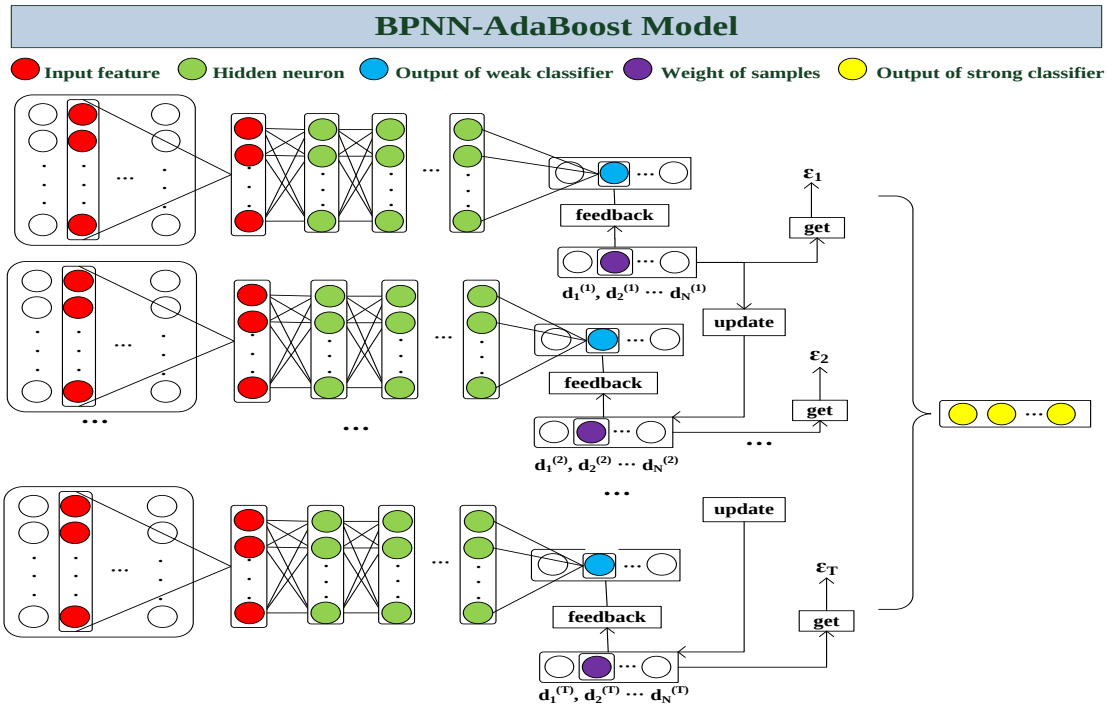

Figure 1. Figure of BPNN-AdaBoost Model

sample weights, we generate a weak classifier by BPNN, then adjust the sample weight by AdaBoost. So that we can get a plurality of such weak classifiers. Finally, we cascade the weak classifiers to get a strong classifier and the final results are obtained by the strong classifier. *Fig. 1* is a diagram of the BPNN-AdaBoost Model.

A. BPNN Algorithm

BPNN is one of the most widely used neural network model, and the model can learn and store a large number of the input-output mapping relationship without prior to reveal the mathematical equations of the mapping relationship. Besides, by using the gradient descent method, the weights of the network are adjusted through the reverse propagation, which makes the sum of square error become minimal. The BPNN model topology consists of the input layer, the hidden layers and the output layer, for hidden layers, the output of neuron j of layer i (i > 0) as show below:

$$x_j^{(i)} = sigmoid\left(\boldsymbol{w}_j^{(i)}\boldsymbol{x}^{(i-1)} + b_j^{(i)}\right) \quad (1)$$

$x_j^{(i)}$ is the output of neuron j of layer i, and $\boldsymbol{x}^{(i-1)}$ is a vector set consisting of the output elements of each neuron in the (i-1) layer. Besides, the 0 layer represents the input vector, which includes 130 features in our model. $\boldsymbol{w}_j^{(i)}$ is the weight vector corresponding to $\boldsymbol{x}^{(i-1)}$of neuron j of layer i, and $b_j^{(i)}$is the bias of this neuron. And the expression form of sigmoid function is:

$$\text{sigmoid(x)} = \frac{1}{1+e^{-x}} \quad (2)$$

If the hidden layers include q layers, the final output in output layer is shown below:

$$o = \text{relu}\left(w^{(q+1)}x^{(q)} + b^{(q+1)}\right) \quad (3)$$

$\boldsymbol{w}^{(q+1)}$ is the weight vector of output layer corresponding to $\mathbf{x}^{(q)}$, and b is the bias of output layer. The function form of relu is :

$$\text{relu(x)} = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \quad (4)$$

Clearly, the output of the result may be a decimal, so we need to approximate them to the nearest output value.

B. BPNN-AdaBoost Model

The core idea of the AdaBoost [6-8] algorithm is to train some different weak classifiers for the same training set, and then put these weak classifiers together to form a stronger classifier. In our model, we use BPNN to generate a weak classifier, then through the AdaBoost algorithm to adjust the weights, so that we can get a plurality of such weak

classifiers. Finally, the weak classifiers are cascaded to get a strong classifier, and the final results are obtained by the strong classifier. The specific application methods are as follows:

- Step 1: We set the number of AdaBoost iteration as T, and if the total number of samples is N, we construct the weight vector ***d*** which correspond to the samples and include N elements. The initial value of the k th element d_k of the vector ***d*** is:

$$d_k^{(1)} = \frac{1}{N} \tag{5}$$

- Step 2: We implement the BPNN algorithm to obtain the estimated value of each sample. And for the t th iteration of AdaBoost algorithm, we construct the error metrics ε_t:

$$\varepsilon_t = \sum_k \text{sign}(o_k^{(t)} \neq y_k)\, d_k \tag{6}$$

$o_k^{(t)}$ stands for the final output of the t th iterations BPNN, which has been rounded, y_k stands the actual output value of the k th samples, and $sign$ is a indicator function that when the condition $o_k^{(t)} \neq y_k$ is satisfied, the output of the function is 1, otherwise, the output is 0. Besides, we set the β_t as the weight of weak classifier:

$$\beta_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) \tag{7}$$

then we update ***d***:

$$d_k^{(t+1)} = \begin{cases} d_k^{(t)} e^{-\beta_t} & \text{if } (o_k^{(t)} \approx y_k) \\ d_k^{(t)} e^{\beta_t} & \text{if} (o_k^{(t)} \neq y_k) \end{cases} \tag{8}$$

- Step 3: We normalize ***d*** according to the linear rule, which satisfies:

$$\sum_{k=1}^{N} d_k^{(t+1)} = 1 \tag{9}$$

And then we repeat Step 2, until the number of AdaBoost iteration T is arrived.

- Step 4: For the *k* th sample, the output of the final strong classifier is:

$$o_k = \sum_{t=1}^{T} \beta_t o_k^{(t)} \tag{10}$$

C. *Parameters Training*

The loss function L of Model training is:

$$L = \frac{1}{2}\sum_{k=1}^{N} d_k (y_k - o_k)^2 \tag{11}$$

Where y_k represents the actual output value of the k th samples, d_k represents the weight of the sample, which is adjusted by the AdaBoost algorithm, and o_k expresses the neural network output value of the k th sample. Obviously, when the network output value and the actual value are consistent better in these samples which have large weights, loss function L will become smaller. We define the set of parameters as θ, which include each weight vector $\boldsymbol{w}^{(i)}$ and bias $b^{(i)}$, and define the α as learning rate, we minimize the L by gradient descent method, the update process can be expressed as:

$$\theta \leftarrow \theta - \alpha \frac{\partial L}{\partial \theta} \tag{12}$$

Because of there are high degree of similarity in the same type Web servers, if we direct identification of the type and version of Web servers, the over fitting phenomenon will occur easily. So we train the model for two steps. First we adopt the feature code as input, and the server type as output, for example, the types of Web servers can divide into Apache, IIS, Nginx, and so on. By the input-output we can effectively pre-train a model that can identify the Web server's type with the parameters of θ, Then, we retain the samples of a particular type model, using the pre-training parameters θ as the initial values, with the samples' corresponding specific version number as output (we only consider the major version and minor version number, without considering the revision number). After the second training processing, we get the final results.

IV. EXPERIMENT

We marked about 2000 Web servers as the experimental samples, and extracted about 130 features for every server. We produced the corresponding input and output through features and Web server's code. In our experiment, 80% samples are selected as training set, and the rest of the samples are used as the test set to verify the correctness of the model. If the Banner information of the Web server is hidden or modified, the identification will fail. The purpose of the algorithm is to identify the Web server that does not depend on the Banner information. So we compare the experiment results with Hmap and Httprint. At the same time, in order to illustrate the effectiveness of the AdaBoost algorithm, we compare the BPNN and BPNN_AdaBoost with the same initial parameters.

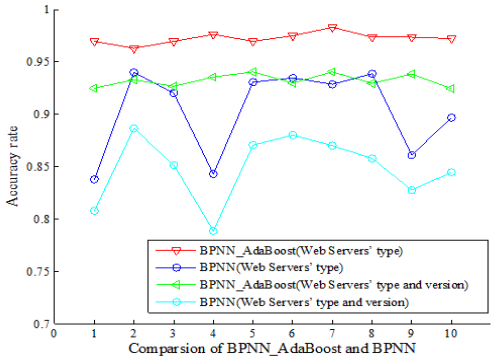

Figure 2. The stability comparsion of BPNN_AdaBoost and BPNN

TABLE I. THE RECOGNITION ACCURATE RATE OF WEB SERVERS' TYPE

	Average accuracy	Apache	Microsoft-IIS	Netscape-Enterprise	Lighttpd	Litespeed	Nginx	Sambar
BPNN_Adaboost	97.34%	99.32%	99.04%	96.20%	95.83%	93.18%	97.37%	95.12%
BPNN	91.61%	93.89%	92.23%	92.94%	91.75%	86.09%	94.21%	88.24%
Httprint	85.22%	87.45%	90.03%	84.12%	—	—	—	—
Hmap	87.12%	88.36%	86.03%	—	—	—	—	—

TABLE II. THE RECOGNITION ACCURATE RATE OF WEB SERVERS' TYPE AND VERSION

	Average accuracy	Apache	Microsoft-IIS	Netscape-Enterprise	Lighttpd	Litespeed	Nginx	Sambar
BPNN_Adaboost	93.03%	95.68%	89.59%	95.50%	95.83%	93.18%	86.57%	95.12%
BPNN	84.84%	91.54%	79.24%	91.88%	91.75%	86.09%	76.40%	88.24%
Httprint	68.89%	68.54%	71.67%	72.23%	—	—	—	—
Hmap	68.35%	69.07%	70.16%	—	—	—	—	—

For the stability test, we have down 10 experiments used with different random initial weights and samples, and the results are shown in Fig. 2. It can be seen that the stability of BPNN_AdaBoost is better than BPNN. Httprint have good recognition results in identifying version 1.3 and version 2.0 of Apache Web Server, but if the version number is higher than 2.0, Httprint will identify it as the version 2.0. For the recognition accurate rate of Microsoft-IIS's type, Httprint have good recognition results, but Httprint will take all the version above 6.0 as 6.0. For Lighttpd, Litespeed and Nginx, Httprint can not identify them, and mistake them as others. Hmap has relatively high recognition rate in Apache, especially in the version of 1.3. For Microsoft-IIS, the recognition accurate rate is just so-so. Hmap has not feature library for Netscape-Enterprise, Lighttpd, Litespeed and Nginx, so it can not identify them. The detail results of recognition accurate rate of Web servers' type are shown in TABLE I, and the detail results of recognition accurate rate of Web servers' type and version are shown in TABLE II.

V. CONCLUSION

Experimental results show that the BPNN-AdaBoost algorithm has very high recognition accuracy in Web servers' type, and the ability of identifying Web servers' type and version is superior to other methods. Recognition accuracy of BPNN-AdaBoost algorithm is slightly higher than the BPNN algorithm, but the stability is significantly better than the BPNN algorithm. By extracting the characteristics of the Web server and using the BPNN-AdaBoost algorithm to identify Web servers, we not only avoid the dependence on Banner information, but also reduce the cost to maintain feature library. More importantly, it improves the recognition accuracy of identify Web servers. In future work, we will strengthen the training and recognition of non mainstream Web servers' model.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (Grant No.61170295), the Project on the Integration of Industry, Education and Research of Aviation Industry Corporation of China under Grant No. CXY2011BH07, and the Co-Funding Project of Beijing Municipal Education Commission under Grant No.JD100060630.

REFERENCES

[1] Durumeric Z, Adrian D, Mirian A, et al. A Search Engine Backed by Internet-Wide Scanning[C]// The, ACM Sigsac Conference. 2015:542-553.

[2] Lee, Dustin, et al. "Detecting and defending against Web-server fingerprinting."[J]. Computer Security Applications Conference, 2002. Proceedings. 18th Annual. IEEE, 2002.

[3] Yang Kexin,Ju Jiubin, Hu Liang. Web service mapping tool WMS[J]. Journal of Jilin University, 2004, 42(2):234-237.

[4] R. Fielding, J. Gettys, J. Mogul, H.Frystyk, and T.Berners-Lee, *Hypertext Transfer Protocol—HTTP/1.1*[S].RFC 2068, IETF, 1997-01

[5] R. Fielding, J. Gettys, J. C. Mogul, H.Frystyk, L. Masinter, P.Leach, and T.Berners-Lee, *Hypertext Transfer protocol—HTTP/1.1*[S], RFC2616, IETF, 1999-06

[6] Bauer E, Kohavi R. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants[J]. Machine Learning, 1999, 36(1-2):105-139.

[7] Sun J, Liao B, Li H. AdaBoost and Bagging Ensemble Approaches with Neural Network as Base Learner for Financial Distress Prediction of Chinese Construction and Real Estate Companies[J]. Recent Patents on Computer Science, 2013, 6(1):47-59.

[8] Sun J, Liao B, Li H. AdaBoost and Bagging Ensemble Approaches with Neural Network as Base Learner for Financial Distress Prediction of Chinese Construction and Real Estate Companies[J]. Recent Patents on Computer Science, 2013, 6(1):47-59.