DeepWAF: Detecting Web Attacks Based on CNN and LSTM Models

Xiaohui Kuang[1], Ming Zhang[1](✉), Hu Li[1], Gang Zhao[1], Huayang Cao[1], Zhendong Wu[1], and Xianmin Wang[2]

[1] National Key Laboratory of Science and Technology on Information System Security, Beijing, China
zm_stiss@163.com
[2] School of Computer Science, Guangzhou University, Guangzhou, China

Abstract. The increasing popularity of web applications makes the web a main venue for attackers engaging in a myriad of cybercrimes. With large quantities of information processing and sharing by web applications, the situation for web attack detection or prevention becomes increasingly severe. We present a prototype implementation called DeepWAF to detect web attacks based on deep learning techniques. We systematically discuss the approach for effective use of the currently popular CNN and LSTM models, and their combinational models CNN-LSTM and LSTM-CNN. The experimental results on the dataset of HTTP DATASET CSIC 2010 demonstrate that our proposed four types of detection models all achieve satisfactory results, with the detection rate of approximately 95% and the false alarm rate of approximately 2%. We also carried out case studies to analyze the causes of false negatives and false positives, which can be used for further improvements. Our work further illustrates that machine learning has a promising application prospect in the field of web attack detection.

Keywords: Web attacks · CNN · LSTM · Detection models

1 Introduction

The web is the abbreviation for the World Wide Web, which plays a central role in the development of the Information Age and has become the primary tool for billions of people to interact on the Internet. Currently, the majority of services on the Internet are provided by web applications with a myriad of information, entertainment, education, commercial and governmental utilities. However, the web security situation is not optimistic. For cyber-criminals, the web has become a main venue for spreading malware and launching cyber-attacks, thus engaging in a wide range of cybercrimes, including information theft, fraud, espionage and blackmail. As early as 2008, Symantec [1] observed that attackers tended to adopt stealthier and more focused techniques targeting computers through the web instead of trying to penetrate networks with high-volume broadcast attacks, and the web-based vulnerabilities had outnumbered traditional computer security concerns with the majority of effective malicious

https://doi.org/10.1007/978-3-030-37352-8_11

activities targeting the web. According to Trustwave [2], hackers are increasingly focusing on and succeeding with application layer attacks.

Among the numerous web security protection solutions, the web application firewall (WAF) is a type of application firewall that applies specifically to web applications. By inspecting HTTP traffic, it can prevent attacks stemming from web application security flaws, such as SQL injection [3], cross-site scripting (XSS) [4], and path traversal [5]. However, the current WAFs typically work in a rule-based mode and rely highly on signatures to detect and prevent attacks. They must have enough characterization and generalization ability to cover normal or malicious behaviors, whereas in practice it is a time-consuming and labor-intensive task to update rules against new emerging attacks. Notably, the renaissance of machine learning, especially the rise of deep learning provides us with new ideas for solving problems. We can build a mathematical model based on sample data to make predictions or decisions without using explicit instructions. Inspired by this, we explore and study how to use deep learning techniques to design a novel and effective WAF—DeepWAF. In this paper, we systematically discuss the approach for using two currently popular deep learning models, namely, convolutional neural network (CNN) and long short-term memory (LSTM), to build web attack detection models.

The rest of the paper is organized as follows. The related work is introduced in Sect. 2. The details of DeepWAF are described in Sect. 3. Experimental results and discussions are presented in Sect. 4. Finally, Sect. 5 concludes the paper.

2 Related Work

Considerable web attacks detection or prevention research [6, 7] has been proposed. Such research ranges from narrow solutions used to prevent only some specific attacks, to generic methods aiming to provide comprehensive protection for web applications.

SQL injections are one of the most common web attacks; thus, a large number of protection methods are proposed specifically to circumvent SQL injection attacks [8–12]. Kar et al. [13] presented an approach for detecting SQL injection attacks by modeling SQL queries as a graph of tokens and using the centrality measure of nodes to train a support vector machine (SVM).

XSS attacks are a type of injection attack in which malicious scripts are injected into the targeted website. Gupta et al. discussed a detailed comprehensive analysis of the exploitation, detection and prevention mechanisms of XSS attacks in [14]. XSS attacks are generally categorized into two categories: stored and reflected. The stored attacks usually rely on client protections to monitor the outgoing HTTP responses [15]. The reflected attacks are generally circumvented by user input sanitizing [16, 17].

HTTP parameter pollution is a special type of attack that supplies multiple HTTP parameters with the same name and may cause a web application to interpret values in unanticipated ways, thus allowing it to be exploited to bypass input validation, trigger application errors or modify internal variable values. Balduzzi et al. [18] presented an automated approach for the discovery of HTTP parameter pollution vulnerabilities in web applications to prevent attackers from compromising application logic to perform attacks.

Protection techniques against other types of web attacks have also been explored. For example, Su and Wassermann [19] proposed a method for preventing command injection based on context-free grammar and compiler parsing techniques. Tajbakhsh and Bagherzadeh [20] presented a framework for preventing local file inclusion attacks. Han [21] introduced a system to detect directory traversal attacks by analyzing web server logs. Saxe and Berlin [22] used a character-level CNN to detect malicious URLs, file paths and registry keys.

Unlike the above work, some research has concentrated on uniform solutions to detect or prevent many types of attacks. Kruegel et al. [23, 24] presented a multi-model approach to detect web-based attacks. They built many statistical detection models on different features, including attribute length, attribute character distribution, attribute order, and access frequency. Corona et al. proposed a formulation of query analysis through hidden Markov models (HMM) to detect attacks on web applications in [25], and presented SuStorID in [26, 27], which is a multiple classifier system that can model legitimate inputs towards web services. Zolotukhin et al. [28] considered analyzing HTTP logs to detect web attacks and employed support vector data description (SVDD), K-means and density-based spatial clustering of Applications with Noise (DBSCAN) to model normal user behaviors.

Choras and Kozik [29] proposed a model consisting of patterns obtained using graph-based segmentation techniques and dynamic programming based on information from HTTP requests to detect cyberattacks on web applications. Bronte et al. [30] proposed an anomaly detection approach that utilizes three measures: cross-entropy for parameters, value and data type, which are intended to compare the deviation between learned request profiles and a new web request. Zhang et al. [31] designed a CNN model to detect web attacks, and the experimental results showed that the model achieves satisfactory results with a high detection rate and a low false alarm rate.

The above work has made great achievements, but only a few have tried to develop protection solutions by using machine learning techniques. Defending web applications is very difficult because there are so many and different attacks. It is necessary to use machine learning, especially deep learning techniques to develop effective protection solutions that are easily implementable and capable of learning. In this paper, we systematically present how to apply two currently popular deep learning models, i.e., CNN and LSTM, and their combinational models to the detection of web attacks.

3 DeepWAF

In this section, we describe the details of DeepWAF. First, the architecture of DeepWAF is introduced. Second, the HTTP request preprocessing algorithm is described. Finally, the four types of detection models, i.e., CNN, LSTM, CNN-LSTM and LSTM-CNN, are presented.

3.1 Architecture of DeepWAF

Figure 1 shows the architecture of DeepWAF with the main focus on the detection phase. Because DeepWAF is a machine learning-based detection system, it must be trained with real web requests before deployment in a real environment to provide protection for web applications. In practical use, DeepWAF can be deployed inline as a reverse proxy.

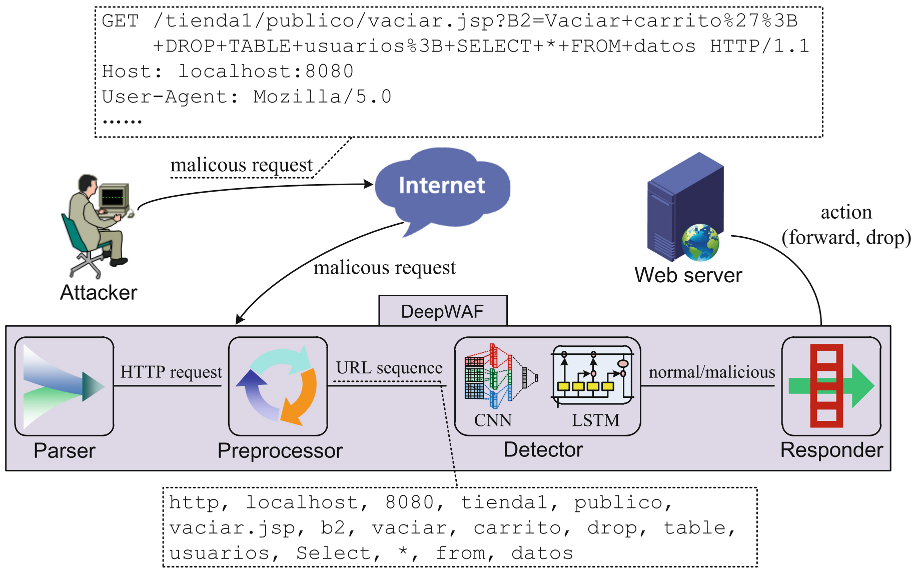

GET /tienda1/publico/vaciar.jsp?B2=Vaciar+carrito%27%3B
+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0
......
malicous request
Internet
Attacker
malicous request
Web server
action
(forward, drop)
DeepWAF
HTTP request
URL sequence
normal/malicious
CNN
LSTM
Parser
Preprocessor
Detector
Responder
http, localhost, 8080, tienda1, publico,
vaciar.jsp, b2, vaciar, carrito, drop, table,
usuarios, Select, *, from, datos

Fig. 1. Architecture of DeepWAF.

DeepWAF is composed of four modules: parser, preprocessor, detector and responder. The typical process for DeepWAF to detect a malicious web request is as follows. First, the request to the web server is parsed and analyzed by the parser into HTTP headers and body. Next, the preprocessor preprocesses the HTTP request and generates a URL sequence that can be fed to the detector. Then the detector detects whether the request is normal or malicious based on the built-in deep learning models. Finally, the responder performs suitable actions according to the detection results. For example, it can forward the request to the web server if the detection result is normal but may drop it if malicious. Since the parser and the responder are similar to those in ordinary WAFs, the following will focus on the implementation of the preprocessor and the detector.

3.2 Preprocessing the HTTP Request

Web attacks exclusively leverage the HTTP protocol to perform malicious activities. If a web server is attacked, that means it receives one or more malicious HTTP requests. Based on this, DeepWAF is designed by inspecting HTTP requests to detect the server-side web attacks. Like other WAFs, DeepWAF can also support the HTTPS protocol by copying the private key used by the server.

The following snippet shows a GET HTTP request from the dataset HTTP DATASET CSIC 2010 [32]. An HTTP request consists of a request line, several request headers and an optional message body (for the POST request). The request line is composed of three components: the HTTP-method, the HTTP-URL and the HTTP-version. Because the vast majority of web attacks are implemented by manipulating the HTTP-URL, and the dataset used in our experiments only contain attacks in HTTP-URL, we focus the detection object on the HTTP-URL. However, without loss of generality, our detection method can be applied to other fields of the HTTP request. A special case is that the POST request contains a message body that can be exploited by injection attacks. So for the POST request, the detection object is defined as the combination of the HTTP-URL and the HTTP-body. For convenience, the detection object is simply called URL in later sections.

```
GET http://localhost:8080/tienda1/publico/vaciar.jsp?B2=Vaciar
    +carrito%27%3B+DROP+TABLE+usuarios%3B+SELECT+*+FROM+datos HTTP/1.1
User-Agent: Mozilla/5.0
Pragma: no-cache
Cache-control: no-cache
Accept: text/xml,application/xml;q=0.9,text/plain
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: localhost:8080
Cookie: JSESSIONID=11F98280E08EE19274786F4EDDDC821F
Connection: close
```

Algorithm 1 Preprocess the HTTP request

```
Input: HTTP request Request
Output: URL sequence url_seq
function PREPROCESSREQUEST(Request)
    if Request.method == "POST" then
        url ← Request.url + Request.body
    else
        url ← Requst.url
    end if
    url ← DECODE(url)
    url ← LOWERCASE(url)
    url_seq ← SPLIT(url, "/, ?, &, =, +")
    return url_seq
end function
```

The procedure of the HTTP request preprocessing, which is used to process the HTTP request into a URL sequence that can be fed to the detector, is shown in Algorithm 1. The main steps are Decode, Lowercase and Split. Since the HTTP URL allows users to encode special characters, attackers often leverage the encodings to hide attack payloads. To effectively detect web attacks, the URL should be decoded first. Because the URL is not case-insensitive, we lowercase all the characters in URL, which can reduce the size of the training vocabulary. The URL is finally split into a sequence by special characters "/", "?", "&", "=", "+", etc. In practice, the preprocessing may be continuously optimized according to the detection results.

For the above HTTP request, one result of the preprocessing is as follows.

```
http, localhost, 8080, tienda1, publico, vaciar.jsp, b2, vaciar, carrito,
drop, table, usuarios, select, *, from, datos
```

3.3 CNN- and LSTM-Based Detection Models

CNN Model. CNN was initially designed for image recognition but has become a versatile model used for a wide array of tasks. CNN can recognize local or high-order structural features of the input. For example, in our detection model, CNN might be able to distinguish that a request containing the words "*table*", "*select*", "*from*", etc. is malicious. The architecture of the CNN-based detection model is shown in Fig. 2. The one-hot encodings X of the URL sequence are input to the embedding layer. The embedding vectors E are convolved on the Convolutional layer with different types of filters, i.e., if the size of E is $l \times k$, the filter sizes are set to $s \times k$ ($s = 3, 4, 5\ldots$), with k equaling the embedding dimension and s taking different values. The max-pooling (over time) takes the largest element from each feature map output by the convolutional layer, and then concatenates them to pass to the Softmax layer. The Softmax layer outputs a label "0" or "1", which indicates whether the request is normal (by label "0") or malicious (by label "1").

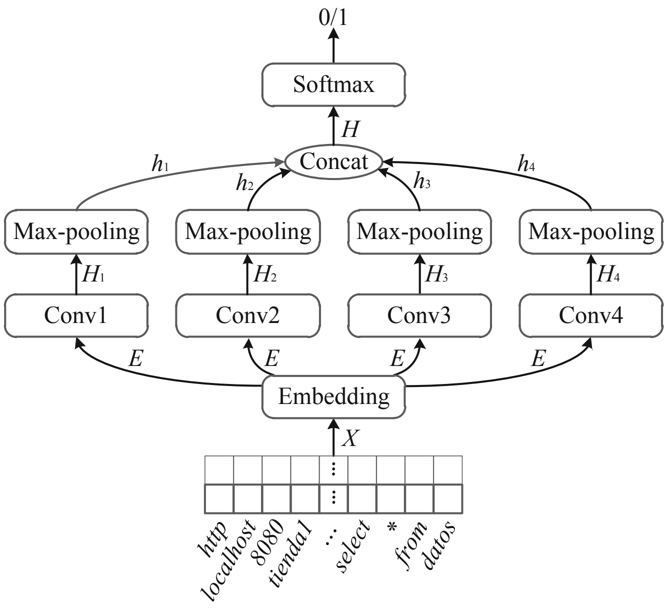


Fig. 2. CNN-based detection model.

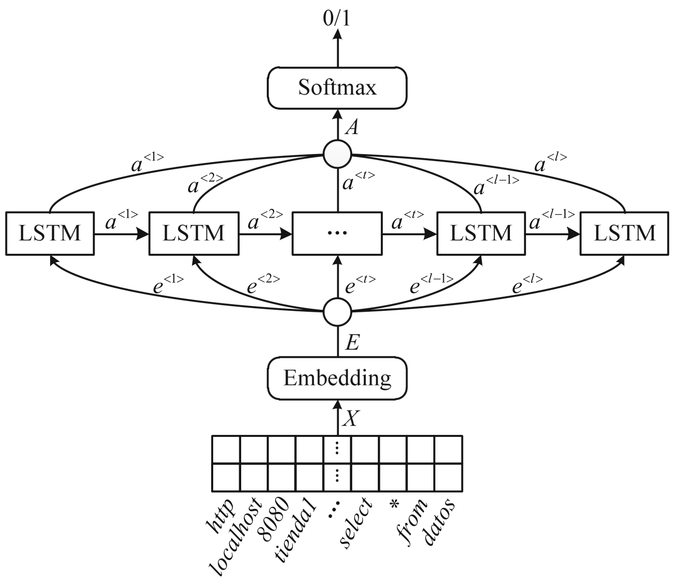


Fig. 3. LSTM-based detection model.

LSTM Model. LSTM is a variant of the recurrent neural network (RNN), which has been proven to perform extremely well on sequential data. In our detection model, LSTM might be able to remember that the word "*from*" appearing in a malicious URL sequence usually follows the word "*select*". The architecture of the LSTM-based detection model is shown in Fig. 3. The length of the time steps is the same as the length of the URL sequence. The embedding vectors of the one-hot encodings are sequentially distributed to different LSTM units. Then, the outputs of all the LSTM units are gathered together to be input to the Softmax layer.

CNN-LSTM Model. The CNN-LSTM model is a combination of CNN and LSTM. As Fig. 4 shows, the convolutional layer receives the embedding vectors as input. Its output is pooled and then fed to the LSTM layer. The output of the LSTM layer is input to the Softmax layer. The intuition behind the CNN-LSTM model is that the CNN will extract structure features, from which the LSTM will learn the sequential features to classify the input.

LSTM-CNN Model. The LSTM-CNN model is a combination of LSTM and CNN. As Fig. 5 shows, the LSTM layer receives the embedding vectors as input. Its output is directly input to the convolutional layer. The output of the convolutional layer is pooled and then input to the Softmax layer. The intuition behind the LSTM-CNN model is that the LSTM generates new sequential encodings of the input, from which the CNN extracts structural features to classify the input.

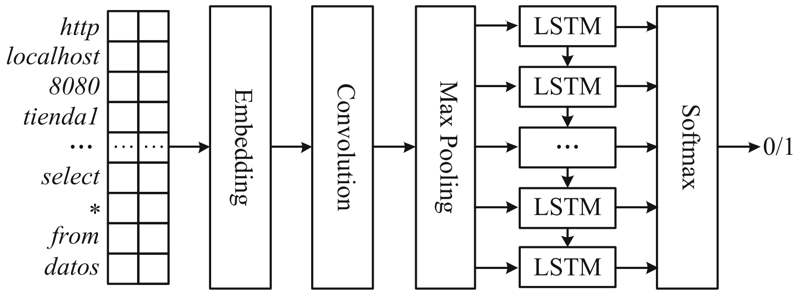


Fig. 4. CNN-LSTM-based detection model.

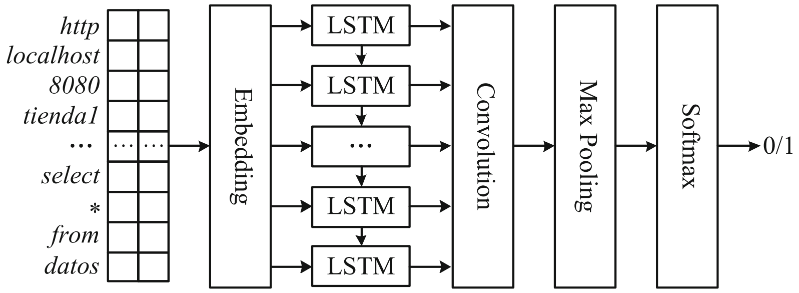


Fig. 5. LSTM-CNN-based detection model.

4 Experiments

To evaluate the performance of models on detecting web attacks, we experimented on the dataset of HTTP DATASET CSIC [32].

4.1 Data Preparation

The HTTP DATASET CSIC 2010 dataset contains thousands of web requests automatically generated by the Information Security Institute of CSIC (Spanish Research National Council), and has been widely used for testing web attack detection systems. The dataset contains 36,000 normal requests and 24,668 malicious requests. The malicious requests include web attacks such as SQL injection, XSS, buffer overflow, information gathering, and file disclosure.

As shown in Table 1, we randomly select approximately 70% of the dataset as training data, approximately 5% as the validation data, and the remaining approximately 25% as the testing data. We train the detection models using the “training data”, tune the parameters using the “validation data” and then test the performance of the detection models on the unseen “testing data”.

Table 1. Experimental data distribution.

	Training	Validation	Testing
Normal	25,200	1,800	9,000
Malicious	17,268	1,233	6,167
Total	42,468	3,033	15,167

4.2 Parameter Settings and Evaluating Criteria

Based on empirical experiences, we set the necessary hyperparameters as Table 2 shows. The embedding dimension is set to 128. The CNN utilizes 4 types of filters with sizes of 3 × 128, 4 × 128, 5 × 128 and 6 × 128. The number of each type of filter is 128. For the LSTM model, the dimensionality of the output space, i.e., the number of hidden units, is set to 64. We train the models by the batch training approach. The learning rate is set to1e-3, and the batch size is 128.

Table 2. Hyperparameter settings.

	Hyperparameter	Value
Embedding	Dimension	128
CNN	Filter sizes	3 × 128, 4 × 128 5 × 128, 6 × 128
	# of filters	128
LSTM	# of hidden units	64
Training	Learning rate	1e−3
	Batch size	128

To evaluate the detection models, we adopted criteria usually used in intrusion detection systems, i.e., *detection rate* and *false alarm rate*, as well as criteria used in machine learning, i.e., *precision*, *recall*, F_1*-measure* and *accuracy*. We use *TP* (*true positive*) to represent the number of malicious requests that are correctly detected as malicious. *FP* (*false positive*) represents the number of normal requests that are incorrectly detected as malicious. *TN* (*true negative*) represents the number of normal requests that are correctly detected as normal. *FN* (*false negative*) represents the number of malicious requests that are incorrectly detected as normal. The evaluation criteria are defined as follows. Note that the *recall* has the same definition as the *detection rate*.

$$Detection\ rate/Recall = \frac{TP}{TP + FN} \tag{1}$$

$$False\ alarm\ rate = \frac{FP}{FP + TN} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$F_1\text{ - }measure = \frac{2 * Precision * Recall}{Precision + Recall} \quad (4)$$

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (5)$$

4.3 Experimental Results

The detection model must first be adequately trained on the training data to perform well on the testing data, i.e., effectively detect web attacks. In practice, the testing data (i.e., the requests to be detected) are unknown to us, so we can only improve the performance of the detection models with training and validation data.

In the experiment, we first observe the model performance on training and validation data, and then adjust the training strategies based on validation accuracy. Finally, we evaluate the detection models on the testing data.

Training Results

There are two commonly used methods to enhance the generalization of the detection model during the training phase, i.e., selecting adequate training epochs and applying dropout. We first simply trained each model for 10 epochs and added dropout after the max-pooling layer with the keeping probability being 0.5, and then performed adjustment depending on the results. The training accuracy and loss were recorded every one step and the validation accuracy and loss were recorded every 100 steps. The results are shown in Fig. 6, where blue curves denote the training metrics and orange curves denote the validation metrics. The CNN, LSTM and LSTM-CNN models exhibit good performance, with accuracy rapidly achieving above 95% and loss decreasing towards 0 on both the training and validation data. The CNN-LSTM model may not seem ideal. It fits the training data well but has a large generation error on the validation data. It also demonstrates that 10 epochs of training are sufficient for these models to achieve stable performance.

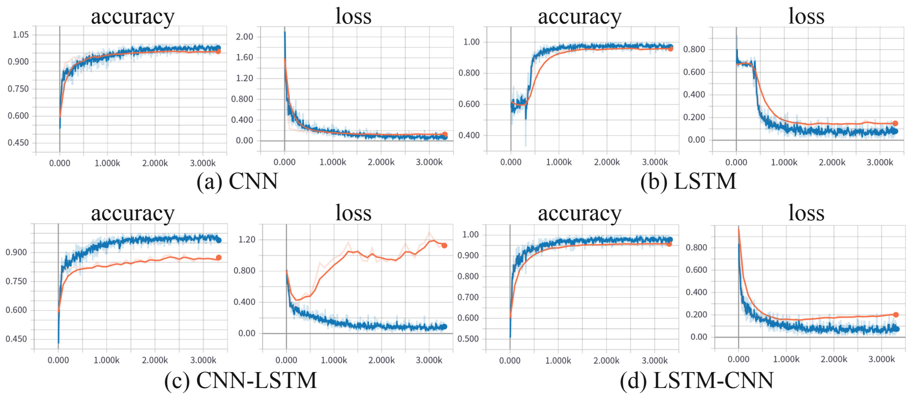


Fig. 6. Training results of the four types of detection models.

Effects of Dropout

In this part, we test the effects of dropout. Dropout has a tunable hyperparameter *p* (the probability of retaining a neuron in the network, or called the keeping probability). A small *p* indicates that very few neurons work during training, and "$p = 1$" means no adoption of dropout. We added dropout after the max-pooling layers and trained the models with different keeping probabilities. The results are shown in Table 3. Since the LSTM model does not contain a max-pooling layer and no dropout is applied, its validation accuracy is always 96.11%. For the CNN and LSTM-CNN models, the dropout provides a very limited contribution to improving the model performance. The validation accuracy varies little with *p*. However, for the CNN-LSTM model, the dropout has a significant negative impact on the validation accuracy. It increases the generalization error. As long as the dropout exists, whatever value the keeping probability takes (i.e., $p = 0.2$, 0.5 or 0.8), the validation accuracy is significantly smaller than that without dropout (i.e., $p = 1$), Which also explains why the CNN-LSTM model does not behave as expected as other models in Fig. 6, where all the models were trained with dropout of the keeping probability being 0.5.

Table 3. Effects of dropout.

Dropout (*p*)	Validation accuracy			
	CNN	LSTM	CNN-LSTM	LSTM-CNN
0.2	0.9608	0.9611	**0.5897**	0.9548
0.5	0.9618	0.9611	**0.8912**	0.9574
0.8	0.9710	0.9611	**0.8945**	0.9568
1.0	0.9651	0.9611	**0.9601**	0.9588

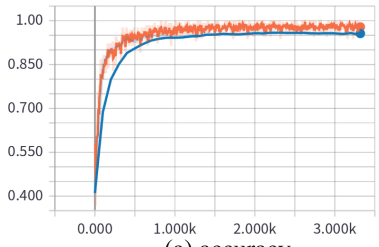
We retrained the CNN-LSTM model without dropout, and the training results are shown in Fig. 7. Obviously, the CNN-LSTM model regains its outstanding performance on both training and validation data.

We think that the aforementioned dropout is improper for the CNN-LSTM model. The dropout is added after the max-pooling layer and before the LSTM layer. It randomly drops some neurons at training time, which is disastrous for the LSTM. The LSTM is learned by sequential information, some of which is unfortunately removed by the dropout. We can conclude that if the CNN and LSTM are sequentially combined to form a CNN-LSTM model, it is not appropriate to apply the dropout before the LSTM, which will undermine LSTM's learning process.

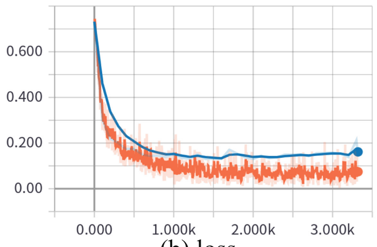
Given the above results, the four types of detection models (i.e., CNN, LSTM, CNN-LSTM and LSTM-CNN) are all trained for 10 epochs without dropout.

Detection Results

After completing the training, we ran the trained models on testing data to evaluate their performance on detecting web attacks. The detection results are as shown in Table 4. In terms of intrusion detection evaluation criteria, each detection model achieves both a high *detection rate* (average approximately 95%) and a low *false alarm*



(a) accuracy



(b) loss

Fig. 7. Training results of CNN-LSTM without dropout.

rate (average approximately 2%). In terms of machine learning evaluation criteria, every model achieves satisfactory performance with high *precision* (average 96.92%), *recall* (average 94.27%), F_1-*measure* (average 95.57%) and *accuracy* (average 96.44%). Because the numerical difference in each criterion is very small (approximately 1–2%), it is hard to determine which model is the best.

Table 4. Detection results.

	CNN	LSTM	CNN-LSTM	LSTM-CNN	Average
Detection rate (*recall*)	0.9549	0.9372	0.9428	0.9359	0.9427
False alarm rate	0.0153	0.0077	0.0367	0.0231	0.0207
Precision	0.9771	0.9882	0.9463	0.9652	0.9692
F_1-*measure*	0.9659	0.9621	0.9445	0.9504	0.9557
Accuracy	0.9726	0.9699	0.9550	0.9602	0.9644

All the models achieved satisfactory detection results, which were obtained just by using the basic CNN and LSTM models with little hyperparameter tuning. Theoretically, the detection results will be better if we adopt more optimal hyperparameter values. Obviously, the results demonstrate that machine learning has great potential to be applied in the field of web attack detection.

Discussions and Case Studies

In this subsection, we provide an intuitive grasp of the number of false negatives (*FN*) and the number of false positives (*FP*), and carry out case studies to explain why some requests are incorrectly detected.

As stated above, the testing dataset contains 9,000 normal requests and 6,167 malicious requests. The *FN* and *FP* of different detection models are shown in Table 5, where "COM" represents the number of requests that are incorrectly detected by all four types of models. Specifically, the same 233 malicious requests are incorrectly reported as normal, and 21 normal requests are incorrectly reported as malicious, which demonstrates that these detection models are more likely to produce the same false negatives but different false positives. Theoretically, if we construct an ensemble model with these four types of models, the *detection rate* can be increased to 96.22% (i.e., 233/6,167), and the *false alarm rate* can be decreased to 0.23% (i.e., 21/9,000), but that will be time consuming.

Table 5. *FN* and *FP* of different models.

	CNN	LSTM	CNN-LSTM	LSTM-CNN	COM
FN	278	387	353	395	**233**
FP	138	69	330	208	**21**

We choose a false negative and a false positive for case studies. The following snippets show two requests. The upper is a malicious testing request that is incorrectly detected as normal and the below is a normal request in the training data. We can see that the following two requests are very similar except that the upper request contains a "%2F", which is the encoding of "/". In our preprocessing algorithm, "/" is regarded as a special character used to split the URL and will not appear in the URL sequence. In other words, the following two requests have the same type of URL sequence after the preprocessing, which explains the reason why the upper request is incorrectly detected as normal.

A "malicious" testing request incorrectly detected as "normal":
```
1: GET http://localhost:8080/tienda1/publico/anadir.jsp?id=3&nombre=Vino
   +Rioja&precio=100%2F&cantidad=55&B1=A%F1adir+al+carrito HTTP/1.1
2: User-Agent: Mozilla/5.0
3: Pragma: no-cache
   ……
```

A "normal" training request:
```
1: GET http://localhost:8080/tienda1/publico/anadir.jsp?id=3&nombre=Vino
   +Rioja&precio=100&cantidad=55&B1=A%F1adir+al+carrito HTTP/1.1
2: User-Agent: Mozilla/5.0
3: Pragma: no-cache
   ……
```

The following snippet shows a normal testing request that is detected as malicious. Through analysis, we find that the following request contains some strings such as "pasar", "por" and "caja", which never appear in the training vocabulary. Such types of requests are very likely to be detected as malicious by the detection models.

A "normal" testing request incorrectly detected as "malicious":
```
1: GET http://localhost:8080/tienda1/publico/pagar.jsp?modo=insertar
       &Precio=6505&B1=Pasar+por+caja HTTP/1.1
2: User-Agent: Mozilla/5.0
3: Pragma: no-cache
   ……
```

The above case studies can be used for further improvements, which we leave as future work.

5 Conclusion

We present a novel web application firewall called DeepWAF by using deep learning techniques to detect web attacks. We first described the architecture of DeepWAF. Then we provided detailed explanations of the HTTP request preprocessing and the principles of the proposed four types of detection models based on CNN, LSTM, CNN-LSTM and LSTM-CNN. Finally, we evaluated the detection models on the dataset of HTTP DATASET CSIC 2010 and verified their good performance in detecting web attacks.

We simply tried the basic CNN and LSTM models with little hyperparameter tuning. Future work can be concentrated on adopting more sophisticated deep learning models, tuning model hyperparameters and inspecting all the fields of the HTTP request, thus resulting in much more powerful web attack detection models.

References

1. Symantec Corporation: Symantec internet security threat report, Trends for July–December 07 (2008)
2. Trustwave: Cenzic application vulnerability trends 2014 (2014)
3. Halfond, W.G.J., Viegas, J., Orso, A.: A classification of SQL injection attacks and countermeasures. In: Proceedings of the IEEE International Symposium on Secure Software Engineering, pp. 13–15. IEEE (2006)
4. Kieyzun, A., Guo, P.J., Jayaraman, K., Ernst, M.D.: Automatic creation of SQL injection and cross-site scripting attacks. In: Proceedings of the 31st International Conference on Software Engineering, pp. 199–209. IEEE Computer Society (2009)
5. Li, H.-F., Lee, S.-Y., Shan, M.-K.: DSM-PLW: single-pass mining of path traversal patterns over streaming Web click-sequences. Comput. Netw. **50**, 1474–1487 (2006)
6. Jensen, M., Gruschka, N., Herkenhoner, R.: A survey of attacks on web services. Comput. Sci. Res. Dev. **24**, 185 (2009)
7. Prokhorenko, V., Choo, K.-K.R., Ashman, H.: Web application protection techniques: a taxonomy. J. Netw. Comput. Appl. **60**, 95–112 (2016)
8. Valeur, F., Mutz, D., Vigna, G.: A learning-based approach to the detection of SQL attacks. In: Julisch, K., Kruegel, C. (eds.) DIMVA 2005. LNCS, vol. 3548, pp. 123–140. Springer, Heidelberg (2005). https://doi.org/10.1007/11506881_8
9. Halfond, W.G.J., Orso, A.: Preventing SQL injection attacks using AMNESIA. In: Proceedings of the 28th International Conference on Software Engineering, pp. 795–798. ACM (2006)
10. Kemalis, K., Tzouramanis, T.: SQL-IDS: a specification-based approach for SQL-injection detection. In: Proceedings of the 2008 ACM Symposium on Applied Computing, pp. 2153–2158. ACM (2008)
11. Liu, A., Yuan, Y., Wijesekera, D., Stavrou, A.: SQLProb: a proxy-based architecture towards preventing SQL injection attacks. In: Proceedings of the 2009 ACM Symposium on Applied Computing, pp. 2054–2061. ACM (2009)
12. Bisht, P., Madhusudan, P., Venkatakrishnan, V.N.: CANDID: dynamic candidate evaluations for automatic prevention of SQL injection attacks. ACM Trans. Inf. Syst. Secur. **13**, 1–39 (2010)

13. Kar, D., Panigrahi, S., Sundararajan, S.: SQLiGoT: detecting SQL injection attacks using graph of tokens and SVM. Comput. Secur. **60**, 206–225 (2016)
14. Gupta, S., Gupta, B.B.: Cross-site scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. Int. J. Syst. Assur. Eng. Manag. **8**, 512–530 (2017)
15. Nadji, Y., Saxena, P., Song, D.: Document structure integrity: a robust basis for cross-site scripting defense. In: Network & Distributed System Security Symposium (2009)
16. Wassermann, G., Su, Z.: Static detection of cross-site scripting vulnerabilities. In: Proceedings of the 30th International Conference on Software Engineering, pp. 171–180. ACM (2008)
17. Weinberger, J., Saxena, P., Akhawe, D., Finifter, M., Shin, R., Song, D.: A systematic analysis of XSS sanitization in web application frameworks. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 150–171. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23822-2_9
18. Balduzzi, M., Gimenez, C.T., Balzarotti, D., Kirda, E.: Automated discovery of parameter pollution vulnerabilities in web applications. In: Proceedings of the 18th Annual Network and Distributed System Security Symposium, pp. 1–16 (2011)
19. Su, Z., Wassermann, G.: The essence of command injection attacks in web applications. In: Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 372–382. ACM, Charleston (2006)
20. Tajbakhsh, M.S., Bagherzadeh, J.: A sound framework for dynamic prevention of Local File Inclusion. In: 2015 7th Conference on Information and Knowledge Technology (IKT), pp. 1–6 (2015)
21. Han, E.E.: Detection of web application attacks with request length module and regex pattern analysis. In: Zin, T.T., Lin, J.C.-W., Pan, J.-S., Tin, P., Yokota, M. (eds.) GEC 2015. AISC, vol. 388, pp. 157–165. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-23207-2_16
22. Saxe, J., Berlin, K.: eXpose: a character-level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys. arXiv:1702.08568 [cs] (2017)
23. Kruegel, C., Vigna, G.: Anomaly detection of web-based attacks. Presented at the Proceedings of the 10th ACM Conference on Computer and Communications Security (2003)
24. Kruegel, C., Vigna, G., Robertson, W.: A multi-model approach to the detection of web-based attacks. Comput. Netw. **48**, 717–738 (2005)
25. Corona, I., Ariu, D., Giacinto, G.: HMM-Web: a framework for the detection of attacks against web applications. In: 2009 IEEE International Conference on Communications, pp. 1–6. IEEE (2009)
26. Corona, I., Giacinto, G.: Detection of server-side web attacks. In: Proceedings of the First Workshop on Applications of Pattern Analysis, pp. 160–166 (2010)
27. Corona, I., Tronci, R., Giacinto, G.: SuStorID: a multiple classifier system for the protection of web services. In: Proceedings of the 21st International Conference on Pattern Recognition (ICPR 2012), pp. 2375–2378. IEEE (2012)
28. Zolotukhin, M., Hamalainen, T., Kokkonen, T., Siltanen, J.: Analysis of HTTP requests for anomaly detection of web attacks. In: 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, pp. 406–411. IEEE, Dalian (2014)
29. Choras, M., Kozik, R.: Machine learning techniques applied to detect cyber attacks on web applications. Log. J. IGPL **23**, 45–56 (2015)

30. Bronte, R., Shahriar, H., Haddad, H.: Information theoretic anomaly detection framework for web application. In: 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), pp. 394–399. IEEE (2016)
31. Zhang, M., Xu, B., Bai, S., Lu, S., Lin, Z.: A deep learning method to detect web attacks using a specially designed CNN. In: Liu, D., Xie, S., Li, Y., Zhao, D., El-Alfy, E.S. (eds.) ICONIP 2017. LNCS, vol. 10638, pp. 828–836. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70139-4_84
32. Gimenez, C.T., Villegas, A.P., Maranon, G.A.: HTTP dataset CSIC 2010 (2012). http://www.isi.csic.es/dataset/