



Article

Deep-Learning Based Injection Attacks Detection Method for HTTP

Chunhui Zhao , Shuaijie Si, Tengfei Tu , Yijie Shi and Sujuan Qin *

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

* Correspondence: qsujuan@bupt.edu.cn

Abstract: In the context of the new era of high digitization and informatization, the emergence of the internet and artificial intelligence technologies has profoundly changed people's lifestyles. The traditional cyber attack detection has become increasingly weak in the context of the increasingly complex network environment in the new era, and deep learning technology has begun to play a significant role in the field of network security. There are many kinds of attacks against web applications, which are very harmful, including SQL (Structured Query Language) injection, XSS (Cross-Site Scripting), and command injection. Based on the detection of SQL injection and XSS attacks, this paper combines the detection of command injection attacks, which are also very harmful, and proposes a multi-classification detection method for web injection attacks. We extract features in the URL (Uniform Resource Locator) and request body of HTTP (Hyper Text Transfer Protocol) requests and combine deep learning technology to build a multi-classification model for injection attacks. Firstly, aiming at the problem of imbalanced distribution of training samples and low detection accuracy of command injection attack, a sample generation method is proposed. The experimental results show that the proposed method ensures a higher detection rate of command injection attacks and lower false alarms. Secondly, we propose a more expressive feature fusion model, which effectively combines the features extracted by deep learning with the discrete features extracted manually. The experimental results show that the feature fusion model proposed in this work is more effective compared with a single deep learning model. The accuracy of the model is improved by about 1%.

Keywords: cyber attacks; command injection; deep learning; cyber security; feature fusion; sample generation

MSC: 68T07



Citation: Zhao, C.; Si, S.; Tu, T.; Shi, Y.; Qin, S. Deep-Learning Based Injection Attacks Detection Method for HTTP. *Mathematics* **2022**, *10*, 2914. <https://doi.org/10.3390/math10162914>

Academic Editors: Valeriu Beiu and Jakub Nalepa

Received: 22 June 2022

Accepted: 10 August 2022

Published: 13 August 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

As information and digitization are highly processed in society with the rapid development of information science and technology, various emerging technologies have brought more opportunities for the development of the new era. The emerging internet technology represented by artificial intelligence has deeply changed people's styles of life and production. Facing the information revolution brought by emerging internet technology, on the one hand, we should take the opportunities; on the other hand, we are also facing a severe issue—network security. Network security issues are becoming increasingly prominent, and the consequences of network security events are becoming more and more serious. Ensuring the security of web applications has become more and more important. In web applications, due to the lack of security awareness of programmers in various companies and the varying programming level of the team, there are often a large number of vulnerabilities or security risks in the server-side applications that process users' HTTP requests. These vulnerabilities and risks make it possible for attackers to

access a large number of users' personal information and sensitive data in the background of the website, and even control the webserver by constructing or injecting malicious values into parameters or bodies of HTTP requests. Many types of web attacks are realized in this way. For example, the web injection attacks in HTTP include command injection, SQL injection [1], and XSS [2] attacks.

In December 2021, accounts of more than three million users of the U.S.-based Flex-Booker appointment scheduling service have been stolen and traded on hacker forums [3]. In February 2022, Croatian phone carrier A1 Hrvatska disclosed a data breach that has impacted 10% of its customers, roughly 200,000 people [4]. At the same time, Japanese automaker Toyota Motors has announced that it stopped car production operations. The outage was forced by a system failure at one of its suppliers of vital parts, Kojima Industries, which reportedly suffered a cyberattack [5]. The expected impact is a 5% drop in Toyota's monthly production in Japan, which translates to roughly 13,000 units.

According to the report of the IT Governance [6], there are 266 security incidents between January and March 2022, of which 161 were cyber attacks, accounting for 61% of those publicly disclosed. These incidents accounted for 75,099,482 breached records. Therefore, we should also consider its possible undetected security vulnerabilities and threats while web technology is widely used in all walks of life. For the entire security industry, in the face of massive log data, it is difficult for security practitioners to quickly and accurately detect all attacks. The traditional intrusion detection methods based on blacklist and rules are not enough to deal with the flexible and diverse web attacks of attackers. The emergence of artificial intelligence technology has brought new ideas to the field of network security. Deep learning technology is data-driven. Through deeply and abstractly learning of the features of a large number of malicious sample data and normal data, it can more accurately and quickly identify various types of attacks, and greatly improve the ability of intrusion detection and security protection. Therefore, the network security technology based on artificial intelligence has gradually become the research hotspot of researchers.

We mainly focus on the detection of web injection attacks with deep learning. Based on the detection of SQL injection and XSS attacks, we combine the detection of command injection attacks, which are paid less attention to in previous research but cause great harm. According to the research on the most widely used artificial intelligence technology—deep learning—we proposed a new detection method. The method learns the differences between normal and abnormal traffic through the deep learning model. Then, it realizes the multi-classification detection of injection attacks based on deep learning. The main contributions in this paper are as follows.

- Aiming at the unbalanced distribution of training samples, a sample generation method is proposed, which effectively alleviates the overfitting problem of model training caused by missing samples. The experimental results show the effectiveness of this method;
- We propose a feature fusion model with more expressive ability based on deep learning. Compared with other classification methods, it effectively improves the overall detection accuracy, reduces the false alarms, and realizes the multi-classification detection of web injection attacks.

The remainder of this paper is organized as follows. In Section 2, we introduce related work. The system models and the main problem are discussed in Section 3. Section 4 presents the experiments and carry out an evaluation on our method. In Section 5, we provide a discussion of our work. Finally, a conclusion is drawn in Section 6.

2. Related Work

We mainly detect SQL injection [7], XSS, and command injection attacks in Web attacks. The traditional detection of web attacks is mostly based on blacklists and rule matching [8–13], which have strong limitations and poor adaptability. In recent years, some works [14–17] have used machine learning methods to classify and detect malicious Web

traffic, but this method is not effective as a general detection method to specifically detect command injection attacks.

Traditional SQL injection detection methods mainly include static detection [18], dynamic detection [19] and combined detection [12]. Static detection refers to detecting errors and correctness through static source code analysis such as the white box test. Gould et al. [18] developed a code analysis tool called JDBC checker, which can only detect some types of SQL injection but can not prevent them. Wassermann et al. [20] extended the white box test to detect tautological SQL injection. The above two methods can only detect some types of SQL injection attacks.

Dynamic detection refers to validating errors or correctness by performing dynamic penetration tests at run time. Yi et al. [19] designed an analysis model embedded in web applications. They parse SQL statements into SQL syntax tree through taint analysis and then determine whether there is a SQL injection attack. Appiah et al. [8] proposed an improved pattern matching method of SQL injection attack detection framework on the base of signature, which distinguishes between SQL queries and malicious queries by integrating fingerprint identification and pattern matching. Pandurang et al. [11] proposed a mapping model for detecting and preventing SQL injection.

Combined detection is a combination of static detection and dynamic detection to detect and prevent SQL injection attacks. Xiao et al. [12] proposed a method to detect and defend SQL injection based on URL-SQL mapping by analyzing user behavior and the response of SQL execution. They extracted pre-defined URLs and corresponding SQL queries to establish a mapping model between HTTP requests and SQL queries. However, there are many uncertainties when the system performs SQL statements. If the extraction of state variables of web applications is not comprehensive enough, it will lead to false positives.

The above methods have something in common. First, researchers usually extract features based on previous experience and then perform string matching to detect SQL injection attacks. However, these methods cannot deal with the increasingly complex and diverse injection attacks in web requests. In recent years, the detection of SQL injection based on artificial intelligence technology has gradually become a research hotspot. Anamika et al. [21] proposed an SQL injection attack detection method based on the Naive Bayesian algorithm and role-based access control mechanism. They achieved a classification accuracy rate of 93.3% on the dataset. Li Qi et al. [22] proposed a method to detect SQL injection attacks based on the LSTM (Long Short-Term Memory) [23] model. Aiming at the problem that shallow machine learning finds it difficult to extract features manually and easily causes overfitting, they proposed a positive sample generation method to expand the dataset and then solved the problem effectively. Finally, the model achieved a relatively high accuracy rate.

In recent years, researchers have proposed many methods to detect cross-site script vulnerabilities. Gupta et al. [13] proposed a method of detecting code files vulnerable to XSS attacks in web applications through a prediction model based on Text Mining. The method extracts text functions from the source code of a web application using a custom marking process, converts every file into a set of unique text functions with related frequencies, and generates a prediction model based on these features. However, this approach is limited to detecting vulnerable code from the application's source code.

Goswami et al. [12] proposed an XSS attack detection method based on unsupervised attribute clustering. They used a cross-entropy Monte Carlo algorithm for ranking aggregation. The clusters are divided into two categories: malicious script and benign script. The method uses divergence measure to detect XSS vulnerabilities on the client. If it exceeds the threshold, then the request is discarded. Otherwise, the request is forwarded to the proxy for further processing. The method needs an environment deployed on the client and proxy server, which lacks flexibility and interferes with the use of customers.

Vishnu et al. [17] used functions based on common and malicious URLs and JavaScript to classify and detect cross-site scripting attacks using three machine learning algorithms

(SVM, naive Bayes, and J48 decision tree). Similarly, Rathore et al. [16] proposed a method to detect XSS attacks on social networking services (SNS) websites based on machine learning. This method uses ten different classification algorithms of machine learning to classify the dataset into XSS or non-XSS based on three parts: URL, web page, and SNS. However, the characteristic of these methods based on traditional machine learning is that the features are only extracted manually, which has some limitations.

In the field of command injection attack detection, there are few works. ANASTASIOS et al. [24] pointed out in 2019 that although command injection attacks are common and can make a big impact with websites, researchers have paid less attention to this type of code injection before. Therefore, the author proposes an open source tool called commix, which can automatically detect and exploit command injection vulnerabilities in web applications. It does a great help to penetration testers and security researchers for detecting and exploiting command injection vulnerabilities. Although the tool is powerful, there are still some shortcomings: firstly, it can be used to test whether there are command injection vulnerabilities in the target website as a penetration testing tool, but it cannot perform tests in large quantities; Secondly, it cannot detect command injection attacks as a dynamic detection tool. We attempt to automatically extract features and perform a detection of a large number of the attack from the perspective of deep learning. Due to the imbalance of the actual sample distribution, it is easy to produce the problem of overfitting.

3. Web Injection Attacks Detection Based on Feature Fusion

The detection method based on feature fusion can realize multi-classification detection of SQL injection, XSS, and command injection attacks on the Web. Web injection attacks achieve the purpose by constructing and modifying the parameters of Web requests. Therefore, we build a detection model by analyzing the URL and Body content in HTTP requests. In our method, the time series features extracted by the GRU (Gate Recurrent Unit) neural network [25] with the discrete features extracted manually by experts are spliced, and are input to the full connected neural network for classification.

3.1. A Method of Sample Generation

Collecting training samples has always been one of the most important problems in machine learning. We need a sufficient number of samples, and at the same time, the number of positive and negative samples is balanced. As shown in Table 1, we have collected a total of nearly 90,000 command injection malicious samples and more than 150,000 benign samples. The benign samples are mainly collected in the campus network monitoring system.

Table 1. The statistics of dataset.

The Type of Samples	The Number of Samples
Command Injection	89,911
Benign Data	156,090
Total	246,001

If the above samples (Table 1) are preprocessed and used for model training directly, the accuracy of the model will be relatively low, and the false positive rate will be high. The reason is that command injection attacks often occur in some scenarios, such as file uploading on web pages, online watermarking, etc. In these scenarios, the web application takes user input as a parameter to execute system call functions such as *exec* and return the result. Due to the limit source of data (network traffic in Campus), traffic with scene characteristics occupies a very small proportion of benign samples, resulting in the emergence of training overfitting problems.

After in-depth analysis and research on the sample data of command injection attacks, we summarize these malicious sample data into the following two characteristics: (1)

Command injection occurs in some dynamic web pages, and the type names of server-side scripts such as CGI, ASPX, and PHP often appear directly in the beginning part of the URL; (2) The part of command injection often appears at the end of the URL and is concatenated with the Linux operators shown in [24]. Furthermore, these malicious data can be abstractly represented as a normal part of URL + shell command operator + command injection part + rest part (usually empty). According to the statistics of nearly 90,000 existing malicious samples, the data that meet the above characteristics accounts for more than 70% of all sample data. The samples of command injection attacks are shown in Table 2.

Table 2. The sample data of URLs in command injection attacks.

<code>/ShowJd.aspx?id=9201&mid=219&pid='ping -c 5 127.0.0.1'</code>
<code>/cgi-bin/bsguest.cgi?email=x;ls</code>
<code>/public/column/43086?type=4&action=type%20%25systemroot%25%5Cwin%2Eini</code>
<code>/info_156.aspx?itemid=22425&&ls</code>
<code>/login.cgi?cli=aa%20aa%27;wget%20http://128.199.251.119/t.php%27\$</code>
<code>/site/search/4288919?keywords=%0a%20SomeCustomInjectedHeader:injected_by_wvs</code>
<code>/News/info_325.aspx?itemid='cat /etc/hosts'</code>

Given the above situation, this work proposes a corresponding sample generation method. The generated samples are added to the benign samples in a fixed proportion to make the distribution of positive and negative samples more balanced. At the same time, we also perform the relevant comparison experiments. The detailed steps of the sample generation method are as follows:

- Classifying command injection malicious samples according to different types of separators;
- According to different delimiter types, we need to find the position where the delimiter appears for the first time in the malicious sample data of command injection attacks;
- Truncating the data after the separator of the sample, and holding the first half of the sample data left after the truncation, which we can consider a benign sample;
- The samples generated by batch programming are added to the benign samples in a fixed proportion (10%, 20%, 30%, 40% respectively).

For example, we can divide the URL: `/info_156.aspx?itemid=22425&&ls` into two parts: `/info_156.aspx?itemid=22425` and `&&ls` according to the delimiter “&&”. We hold the first half of the URL and use it as the generated sample data.

3.2. Model

3.2.1. Overview

In the existing related works [8,12,17,21,22,26], deep learning and general machine learning have been used widely in the detection of Web injection attacks. Deep learning models are usually built based on the training of a large number of datasets, which is to effectively model and detect data with specific patterns or features. Although the performance of deep learning models is not bad in most cases, there are still some shortages, such as insufficient modeling of attack types for a small amount of sample data.

For example, we can divide SQL injection attacks into four types: union, Boolean, error-based, and wide-byte injection. For common types of SQL injection, such as union injection attacks, it is usually easy to collect data, and the detection results obtained by the model will be sound. However, data collection for uncommon injection types such as wide-byte injection is likely to be difficult. It lacks the training data of these subdivision types, resulting in insufficient training of the deep model for this type of data, which will eventually lead to a high rate of false negatives in the detection of this type. In this situation, the usual practice is to repeatedly sample this small number of sample types or use a sampling algorithm to expand samples. The new samples obtained in this way often have problems such as sample overlap or poor sample quality. Sub-training often has serious over-fitting issues, resulting in insufficient generalization ability.

The features extracted by experts can make up for the above situation. Experts have an in-depth understanding of the classification of SQL injection attacks and the various categories. Experts can often propose better solutions based on their own domain experience and knowledge. Representative features are applied to achieve coverage of specific subdivision types. Discrete features are extracted by experts manually, which can often achieve more complete detection rates and better detection results with high-level features extracted by deep learning models.

The framework of the detection model is shown in Figure 1. The detection model is composed of two parts. One part is sequence feature extraction. The GRU model automatically extracts sequence features from the preprocessed Web attack sample data; The other part is to extract discrete features manually. Experts define some features through a priori knowledge, then extract discrete features from the input data and standardize the features. The extracted sequence features and discrete features are then stitched together and input into the fully connected neural network for classification. The classification results are normalized at the Softmax activation function layer and output.

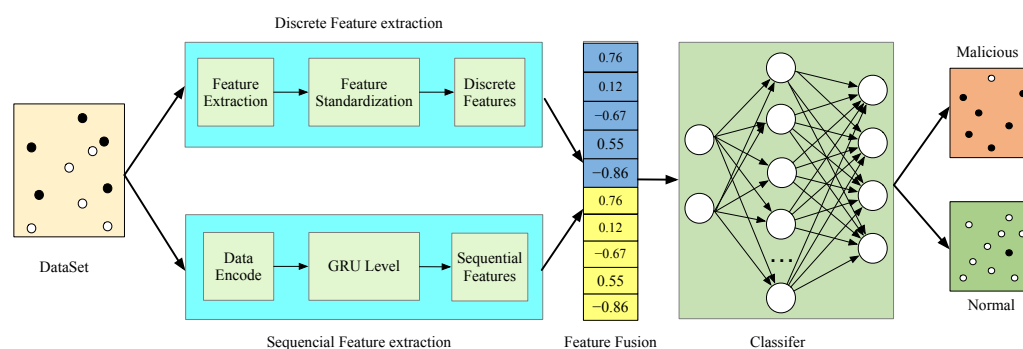


Figure 1. The architecture of model.

3.2.2. Preprocessing

After collecting malicious sample data and benign traffic data, we need to perform preprocessing operations on the dataset, including URL decoding, binary data stream processing, removal of Chinese characters, and interception of fixed lengths.

- URL decoding: URL data are usually encoded to convert the data format. For example, the URL `http://[domain]/all.php?kd=%C8%CB%B9` is converted to `http://[domain]/all.php?kd=com` after URL decoding. A short and clear URL can be obtained by URL decoding, which is convenient for model learning and understanding;
- Binary stream data processing: Considering that there are some situations such as image transmission, file upload and video stream in the traffic, the POST body in HTTP requests will contain some meaningless binary stream data, and this part data is not meaningful for training the model. Therefore, we first identify the starting position of the binary stream data, and then replace it with a predefined short string. The rule of string replacement is shown in Table 3;
- Remove Chinese characters: Chinese characters in the URL are useless to the classification of the detection model and may interfere to a certain extent, so the Chinese characters in the web request traffic data are directly removed in the data preprocessing stage. The specific method is to convert the training data to lowercase, and only keep the data whose ASCII codes are in the range of 33 to 126;
- Keep a fixed length and map to a number interval: We set a fixed length L , truncate the part whose length exceeds L , and add zero before the data whose length is less than L . Next, we map the fixed-length traffic data to the corresponding ASCII code values according to the character sequence, and get the vector data of the input model.

Table 3. The rule of string replacement.

The Type of Strings	The Replacement of Strings
Binary stream data	Binary_Data
Encryption data	Encryption_Data
MD5	MD5_Hash
SHA1	SHA1_Hash

3.2.3. Discrete Features Extraction and Standardization

In the discrete feature extraction section, we mainly extract the following features to carry out related experiments to verify the feasibility of feature fusion methods.

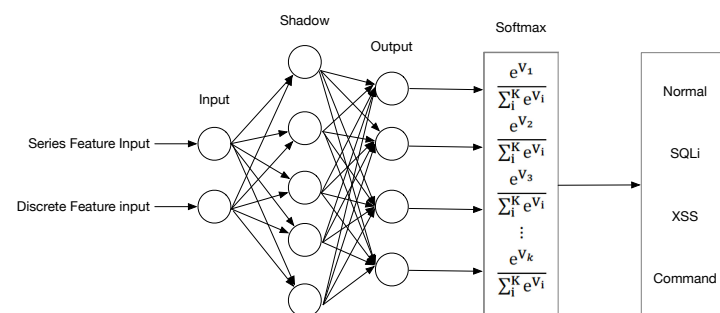
- Keyword features: Keywords are very important for detecting Web injection attacks. For example, although the ‘<script’ keyword rarely appears in URLs, it often arises in XSS attack payloads; Similarly, ‘select’ and other SQL statement keywords which appear in the URL are key features to check for SQL injection attacks;
- Length feature: Web attacks often need to construct some attack payloads, so the length of malicious URLs is statistically different from that of benign URLs;
- Other features: The features of number proportion, special character proportion and whether the parameter value contains IP are also key differences between benign requests and malicious requests.

After extracting artificial discrete features, the next step is to standardize the extracted discrete features. Because the quantitative scale of each feature is not necessarily the same, when training the model, the features with a larger scale will play a decisive role and features with a smaller scale will play a small role. In the feature standardization stage, we control the discrete features of different scales under the same standard scale, eliminate the influence of scale differences and feature units, and speed up the convergence speed of model training. Therefore, the step of discrete feature standardization is very necessary. Feature standardization maps the original discrete features to the distribution with the mean value of 0 and standard deviation of 1. Specifically, assuming that the mean value of the original discrete features is μ and the standard deviation is σ , the standardization formula is defined as Equation (1):

$$z = (x - \mu) / \sigma \quad (1)$$

3.2.4. Classifier

We input the fusion features obtained by series sequence features and discrete features into the classifier for classification. The classifier uses a fully connected neural network and softmax activation function to realize classification and normalization. The structure of the classifier is shown in the Figure 2.

**Figure 2.** The architecture of classifier.

The fully connected neural network (FCN) is the most basic neural network, and its network structure is a typical chain structure. In FCN, every node in the $(n - 1)th$ layer is connected to all the nodes in the $(n)th$ layer, and the data is to read through the input layer.

The hidden layer in the network is a vector type, and each element in the vector is similar to the function of a neuron. The current layer receives the input from the previous layer of neurons and calculates the activation value. The output layer gets the output of the last hidden layer, and then it is used for classification.

There is more than one neuron in the output layer of the FCN, and it can have multiple outputs. It can be flexibly used in practical application scenarios such as clustering and multi-classification. Each neuron learns a linear relationship between input and output. If considering $x_1, x_2, x_3, \dots, x_n$ as the input data, each neuron randomly initializes the weight parameter and a bias parameter. According to $z = \sum_{i=1}^n \omega x_i + b$, we get the calculation result and use the neuron activation function *Softmax* to calculate the output. Finally, we obtain the feature mapping result.

4. Evaluation

In this section, we evaluated the effectiveness of the sample generation method and the detection model of injection attacks. We performed our experiments on a machine with 16 GB of RAM with Intel Core i5-8265U CPU and NVIDIA GTX 1060 GPU. Further, we conducted our work with Pytorch 1.5.0 (Soumith Chintala; New York, NY, USA) and Python 3.8.5 (Yury Selivanov; San Francisco, CA, USA) in Ubuntu 16.04.

4.1. Sample Generation Method

We combine the cutting-edge deep learning technology with network security. Then we use the characteristics and advantages of deep learning to detect the command injection attack, which is very harmful in web attacks. It can cover the shortage of existing detection methods and achieve a large number of real-time detection of command injection attacks. We conduct in-depth analysis and research from the URL and post body of HTTP requests. Then we propose a detection method for command injection attacks. On the one hand, this method considers the characteristics and principles of command injection attacks. On the other hand, it combines the cutting-edge technology of deep learning and provides ideas for web attack detection in the next section. At the same time, we propose a sample generation method to supplement the command injection training dataset. It alleviates the over-fitting problem caused by unbalanced sample distribution in the model training process. This method effectively improves the detection accuracy and reduces the false positive rate.

To solve the problem of over-fitting during model training, caused by the imbalanced distribution of training samples, we propose a sample generation method that balances the sample dataset. Firstly, according to the proportion of 10%, 20%, 30%, and 40%, we make four copies of normal samples and add the generated sample data to each benign sample. We use $DS_i (i = 1, 2, 3, 4, 5)$ to represent the i th dataset. The details of each dataset are shown in Table 4. Here, the number of malicious samples of command injection attacks is 89,911. With the sample generation method (SGM) proposed in this work, the generated samples are added to the benign samples in different proportions, which form five different datasets- $DS_i (i = 1, 2, 3, 4, 5)$.

Table 4. The statistics of Sample Data.

Dataset	Malicious Sample	Normal Data		
		SGM/Total	SGM	Total
DS_1	89,911	0	156,090	0%
DS_2	89,911	15,600	171,690	10%
DS_3	89,911	31,200	187,290	20%
DS_4	89,911	46,800	202,890	30%
DS_5	89,911	62,400	218,490	40%

The five different datasets were divided into a training set, validation set, and test set according to the ratio of 6:2:2. The model uses the training set to continuously adjust its parameters to obtain better classification results. It uses the validation set to try different model hyperparameters during the multiple training process of the model to select the optimal parameter combination. Finally, the model uses the test set to evaluate the classification model and its actual generalization ability. We use deep learning technology to detect command injection attacks and to observe the impact of the sample generation method on the model detection effect, which is essentially a binary classification problem. The set of command injection samples is regarded as “positive”, benign samples are regarded as “negative”, and the accuracy (ACC), precision (PRE), recall (Recall), and F1-value are used as performance evaluation indicators of the model.

Before explaining how these evaluation indicators are calculated, we need to introduce the meaning of several symbols:

- ‘TP’ indicates the number of positive samples predicted to be positive;
- ‘TN’ indicates the number of negative samples predicted as negative;
- ‘FP’ represents the number of negative samples predicted as positive, i.e., “False Positives”;
- ‘FN’ represents the number of positive samples predicted as negative class, i.e., “False Negatives”;

Combined with the application scenarios of the detection model in this work, the actual calculations of these three evaluation indicators are as follows.

The accuracy rate represents the percentage of the data for which all predictions are correct (both positive and negative) to the total number of samples:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \quad (2)$$

The precision rate is for the prediction results of the model, which means that the samples correctly classified as positive classes account for the percentage of all samples that are classified as positive classes by the classified model:

$$\text{Precision} = \frac{TP}{TP + FP}. \quad (3)$$

The recall rate is for the original sample, which means that the samples correctly classified as positive class account for the percentage of all positive class samples in all test data:

$$\text{Recall} = \frac{TP}{TP + FN}. \quad (4)$$

The F1-value is a comprehensive evaluation index that balances recall and precision, and is expressed as the harmonic mean of the recall and precision indices:

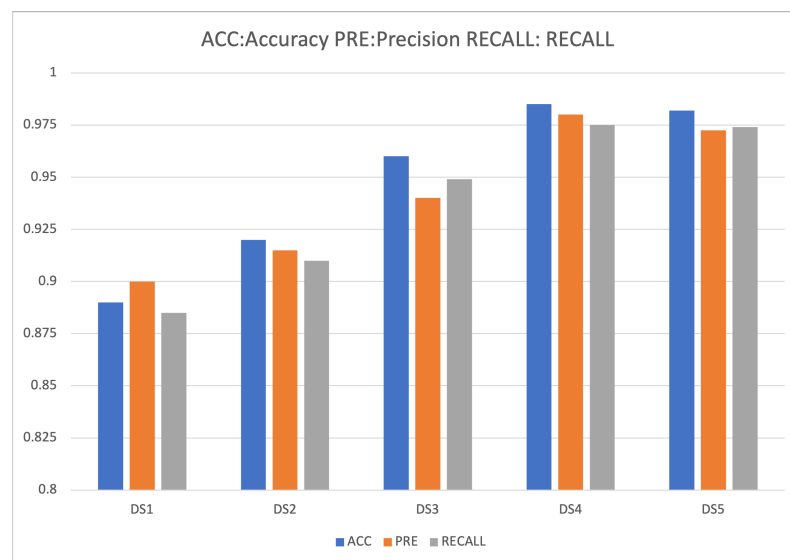
$$\text{F1-value} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (5)$$

In the model training stage, we obtain the optimal hyper-parameters’ configuration through many tests shown in Table 5. We set the initial learning rate to 0.01 and the decline rate of the learning rate to 0.5. The learning rate decreases dynamically every five epochs in the training stage. We use the mini-batch gradient descent as the optimization method of the training model. What is more, we set batch_size to 64 and the number of epochs to 20.

We use the five datasets to train the GRU neural network model. The five neural network models use the same parameter configuration in Table 5 and record the best result of each model. The experimental results are shown in Figure 3.

Table 5. Hyperparameters' setting of the training model.

Parameters	Values
The length of the input data	150
Word embedding dimensions	128
The size of GRU hidden layers	64
GRU layers	2
The size of the fully-connected hidden layers	64
Learning rate	0.01
Epochs	20

**Figure 3.** The experimental results of the indicators on different datasets.

As can be seen from Figure 3, with nearly 90,000 command injection samples and 156,000 benign samples, the accuracy, precision, and recall of the classification model are generally low before the sample generation performed, 89%, 90%, and 88.7%, respectively. After using the sample generation method described in Section 3.1, white samples are generated, and various evaluation indicators improve as the proportion of white samples increases. On the DS_4 dataset, the accuracy, precision, and recall rates reach the highest, at 98.6%, 97.9%, and 97.4%, respectively, and all three indicators exceeded 97%. After increasing the ratio to 50%, the indicators no longer improve but decrease slightly. The experimental results show that the sample generation method proposed can effectively alleviate the problem of overfitting caused by the unbalanced distribution of samples. It enables the classifier to achieve a better generalization ability and higher detection accuracy.

In this part, aiming at the over-fitting problem caused by the unbalanced distribution of positive and negative samples, we propose a sample generation method and use a GRU neural network model for comparative experiments. The experimental results show that it can effectively improve the detection accuracy of command injection and reduce the false positive rate. Next, we conduct experiments to observe the effectiveness of the injection attack detection model, which is a multi-classification model that can detect the three web injection attacks simultaneously.

4.2. The Detection of Injection Attacks

Based on the experimental method in Section 4.1, we propose a detection method of web injection attacks based on feature fusion, which can realize multi-classification detection of SQL injection, XSS and command injection in Web attacks. It is similar to command injection attacks that SQL injection and XSS can also be achieved through constructing the parameters of web requests. Therefore, we consider the URL and request body in the web

request based on HTTP protocol to detect these three types of injection attacks. Our feature fusion-based detection method combines the temporal features extracted by GRU neural network with the discrete features manually extracted by experts. Then, we input the fused features into a fully connected neural network for classification.

4.2.1. Dataset

We mainly detect SQL injection, XSS, and command injection attacks in web injection attacks. Our samples mostly come from the websites' communication traffic extracted from the campus network monitoring system. One part of the training dataset for XSS attacks is from a well-known network security company, and the other part is from the public dataset—HTTP DATASET CSIC2010 [27].

After cleaning and deduplication of the collected data, the dataset required for the experiments is finally obtained, including 202,889 benign samples and more than 170,000 malicious data. We divide the training sample into training set, validation set and test set according to the ratio of 3:1:1. It should be pointed out that the total number of benign samples here is the number including the samples generated by the sample generation method in Section 3.1. The specific distribution of the training set is shown in Table 6.

Table 6. The specific distribution of training set.

The Type of Data	Amount
Normal	202,890
Command Injection	89,911
SQLi	45,078
XSS	40,327
Total	378,206

4.2.2. Results

Our model detects Web injection attacks based on feature fusion. The results of various evaluation indicators on the test set are shown in Table 7. Our purpose is not to classify web request traffic data into benign and malicious, but to detect SQL injection, XSS and command injection attacks in web traffic through a multi-classification model. Therefore, the calculation method of the evaluation index of the multi-class model is different from the two-class problem. We first get the accuracy, precision, recall and F1-value results for each kind of attack. Then we calculate the arithmetic mean of each evaluation metric and use it as the result of the detection model.

Table 7. The result of the model on evaluation indicators of validation set (%).

Accuracy	Precision	Recall	F1-Value
99.39	99.51	99.37	99.43

As we can see from Table 7, the detection model of web injection attacks based on feature fusion has achieved a rate of accuracy of more than 99.3% on the test set of more than 75,000 data. The evaluation indicators of recall, precision, and F1-value rates also have high results. The above results show that the detection model based on feature fusion has strong generalization and can accurately detect web injection attacks in communication traffic.

Figures 4 and 5 show the changes in the accuracy and loss function of the validation set and the training set during ten rounds of training for the feature fusion detection model. We can see from the figures that the convergence of the model parameters is normal.

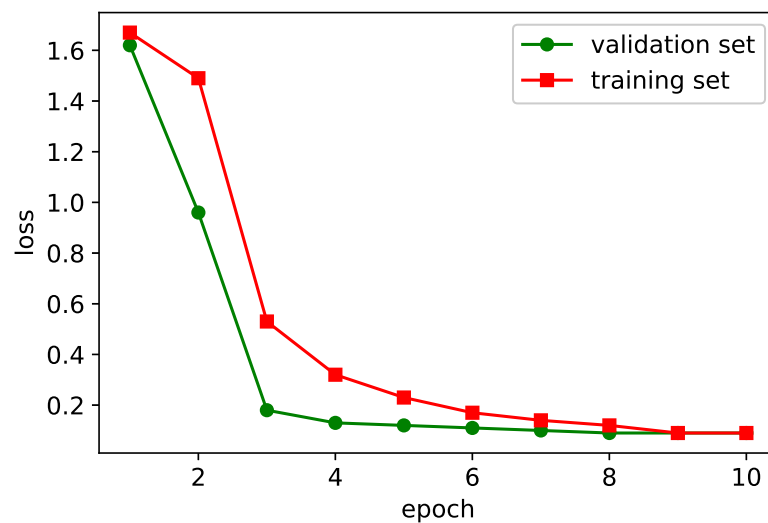


Figure 4. The loss curve of the feature fusion detection model on the validation set and training set.

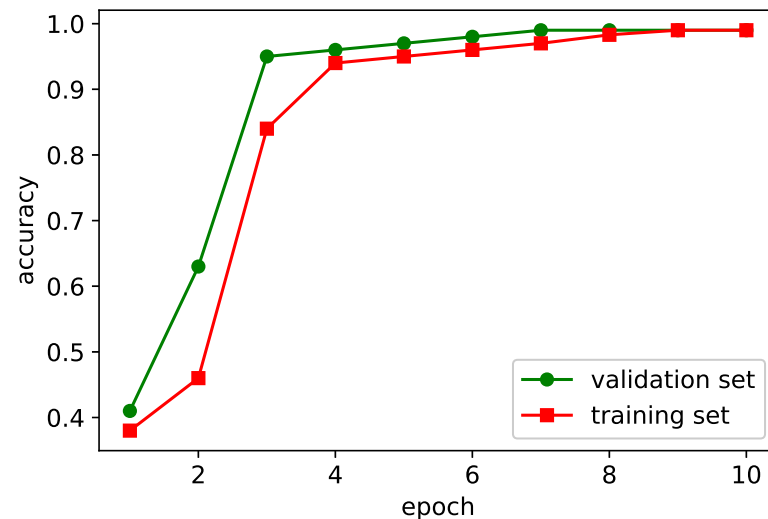


Figure 5. Accuracy curve of feature fusion detection model on validation set and training set.

4.2.3. Comparison

In order to verify the effectiveness of the feature fusion method proposed in this work, we performed relevant comparative experiments.

First, the length of the input affects the detection speed, the training time, and the accuracy of the model detection. Accordingly, it is necessary to find a reasonable value for the input data's length. After the preprocessing steps, the box plot of the length distribution of all training data, including web injection attacks samples and benign traffic data, is shown in Figure 6. The "len" indicates the total length of the URL and the content of the POST body.

According to Figure 6, the median length of the training data is around 100, and the 3/4 quantile is about 150. It indicates that about 75% of the training sample data is less than 150. Therefore, the input lengths of 100, 150, and 200 are used for experimental comparison here to verify the influence on the accuracy and speed of the model with different data lengths. Other settings of the experiment here are the same.

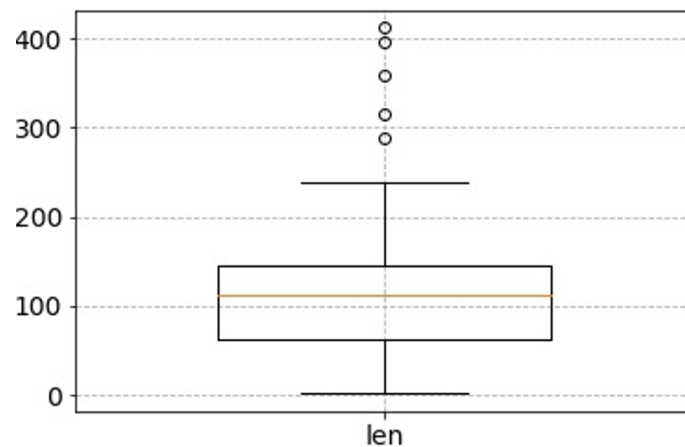


Figure 6. The box-plot of the length distribution.

The results of model detection with different input lengths are in Figure 7. The time required for training models to detect more than 75,000 web request traffic data in the test set based on three different input lengths is in Table 8. We can see that when the input length is 100, the model detection speed is the fastest, but the results of the detection model are generally low, especially since the recall is only about 93%. When the input length is 100, it is not enough for the detection model to capture the features of the input data. When the input length is 150, the values of accuracy (ACC), precision (PRE), and recall (RECALL) have increased. It is about 4% higher than the average result when the input length is 100, and the three evaluation indicators have exceeded 98%. When the input model length is 200, the values of accuracy and recall are slightly improved based on the input length of 150. However, the model detection speed is significantly reduced compared with the input length of 150. Table 8 shows that when the input model data length is 200, the detection time is increased by 235 s compared with the input length of 150, and the detection speed is reduced by about 44%. We choose 150 as the final input length of the model in this work under the consideration of speed and accuracy.

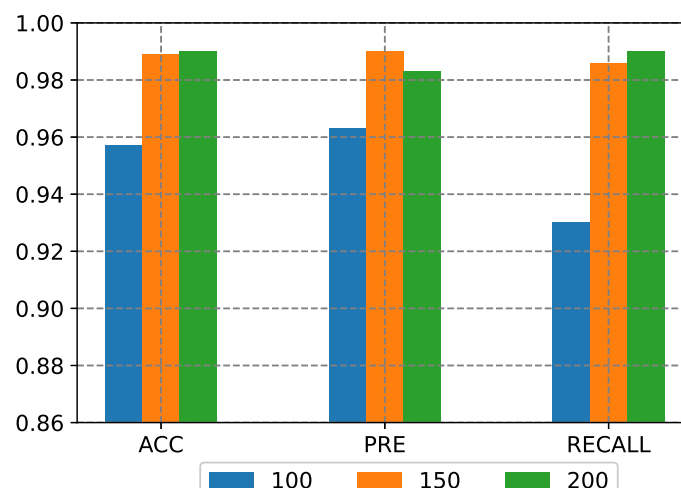


Figure 7. The results of model detection effects with different input lengths.

Table 8. The time required for the model with different lengths of the input.

Length	100	150	200
Time	439.67 s	529.31 s	764.55 s

Then, we compare the model effects of general machine learning methods (such as Logistic Regression, Naive Bayes, and SVM). We train all models with the datasets shown in Table 6. The comparison results are shown in Table 9. It can be seen from the table that in the detection of web injection attacks, although Logistic regression and the Naive Bayes algorithm have relatively reasonable precision values, the results of precision and recall are different from the precision values, which indicates that there is a high rate of false negative in the detection results. It represents that many malicious requests are identified as benign requests, which shows that the two algorithms have not really learned the characteristics that can identify benign and malicious traffic. Compared with the two algorithms, the experimental results of the SVM algorithm are much better. The accuracy rate, precision rate and recall rate all exceed 90% and even the precision rate reaches 95.03%, which shows that the SVM has identified the difference between benign and malicious traffic in a certain extent. The detection model proposed in this paper is superior to the above three general machine learning methods in three performance indicators. The results of the three performance indicators all exceed 99%, which also shows that the deep learning model has certain advantages in detection performance compared with machine learning.

Table 9. The comparison of our model with machine learning methods (%).

Models	Accuracy	Precision	Recall
Logistic	79.81	97.6	69.37
Naive Bayes	85.13	93.27	57.16
SVM	92.79	95.03	93.47
Our method	99.39	99.51	99.37

Then, we compare the multi-class detection method of injection attack based on feature fusion with other neural network classification models. We choose the single model and fusion model as control models. We used the C-LSTM model [28], which is a combination of long short-term memory network and convolutional neural network, in the mixed control model, and the model eXpose [29], which performs convolution and splices based on four different windows, is used as the second mixed control model. The single-model control experiments used two models: Char-GRU and Char-CNN [30].

Char-GRU comes from the char-RNN model [31]. We use GRU [25] to replace the RNN model, which solves the problem of gradient disappearance or gradient explosion of the basic RNN model in long series [32], and the effect of Gru and LSTM is similar but simpler than LSTM [33]. Char-GRU performs character embedding on the sample data, and extracts time series features with the GRU deep learning model, finally uses a fully connected layer for classification. Char-CNN uses a convolutional neural network to automatically extract text features after character embedding on the input sample data, and finally uses a fully connected layer to classify the sample data. Table 10 shows the comparison of the classification effects of the web injection attack of the detection model based on feature fusion and the control model on the test set.

Table 10. The comparison of our model based on feature fusion and other classification models (%).

Models	Accuracy	Precision	Recall	F1-Value
Char-GRU	98.67	98.31	98.06	98.18
Char-CNN	97.38	98.49	96.41	97.43
C-LSTM	99.23	99.36	99.18	99.26
eXpose	98.07	99.13	97.85	98.48
Our model	99.39	99.51	99.37	99.43

To fully represent the detection effect of each control model, we set the dropout parameter of each model to 0.5, and stop when the performance index of the neural network model does not change after five consecutive cycles on the validation set training

to prevent overfitting. The initial learning rate of the model is set to 0.01, and the learning rate decays exponentially every five epochs.

According to Table 10, the overall performance of the deep learning model is pretty good. The performance of the single model is inferior to the fusion control model, mainly because the deep fusion model fully extracts multi-level sample features. The accuracy of the Char-CNN model is higher than that of the Char-GRU, indicating that the false positive rate of the CNN model is lower. However, the accuracy and recall rate of the GRU model are higher than those of the CNN model, indicating that the performance of the GRU model is better than the CNN model in terms of feature mining and learning.

Our proposed feature fusion model outperforms the eXpose model on every metric. Our model guarantees low false negative and false positive rates while maintaining high accuracy. Compared with the eXpose model, the discrete features extracted by us are more complementary to the temporal features automatically extracted by the GRU neural network model. At the same time, it shows that our model can learn the performance characteristics of Web injection attacks well.

The C-LSTM model shows the best performance among all the comparative models, with each index exceeding 99%, and the F1-value reaching 99.26%. The reason C-LSTM can achieve this effect is that it integrates the advantages of Convolutional Neural Networks and Recurrent Neural Networks. The C-LSTM uses the convolutional neural network to obtain the local abstract features, extracts the sequence features with the recurrent neural network, and puts them together to the classifier for classification. However, the C-LSTM model has higher structural complexity compared with our model, so our model outperforms it in training time and memory consumption. In addition, the performance of our model is also better than that of the C-LSTM, because our proposed multi-classification model based on feature fusion adds discrete features derived from expert knowledge based on time series features. To a certain extent, our model reduces the parameter quantity and structural complexity, and ensures the adequacy and complementarity of feature extraction at the same time.

5. Discussion

Through experiments, we effectively improve the detection accuracy of injection attacks and reduce the false positive rate through the sample generation method. Our approach focuses on testing the effect of this method on detection accuracy. In terms of performance and accuracy, we choose to ensure the effectiveness of this method in improving the accuracy of model detection first. Experiments show that different inputs of data length do affect the training efficiency and accuracy. Therefore, we choose the appropriate fixed length input through experiments to ensure that our model can finally obtain better detection accuracy and runtime performance.

Our method conducts the detection of web injection attacks based on feature fusion. We choose the GRU to extract temporal features automatically. Additionally, we integrate discrete features extracted by experts to improve the overall detection accuracy of the model. It solves the problem of false positives caused by insufficient and specific classification data. The C-LSM is more complex than our model with ensuring accuracy, because it combines a convolutional neural network and a recursive neural network. We use the GRU neural network to automatically extract temporal features and combine discrete features defined by experts, which reduces the number of model parameters and structural complexity. It also ensures the sufficiency and complementarity of feature extraction. Compared with other models (char-GRU, char-CNN and eXpose), we obtain a relatively higher detection accuracy.

6. Conclusions

In this paper, we propose a detection method of web injection attacks, which have a stronger feature expression ability and better detection performance. The method makes up

for the shortcomings of traditional attack detection technology and provides a foundation for the wide application of deep learning technology in network security.

We combine artificial intelligence technology with network security. From the perspective of attack behavior, we use the advantages of deep learning to detect injection attacks that are very harmful in web attacks, making up for the shortcomings of previous detection tools.

First, we propose a sample generation method to solve the problem of over-fitting caused by unbalanced samples in the training process of the neural network model, and experiments show that this method can effectively alleviate the over-fitting problem in the training process. The fitting problem improves the accuracy of the detection model. Then, we propose a feature fusion based method to detect SQL injection, XSS and command injection attacks. This method fuses the time series features extracted by the neural network model with the discrete features defined by experts, and uses the fused features to train the classifier. The experimental comparison with other classification models proves the advantages of our proposed detection model in terms of accuracy and generalization ability.

Author Contributions: C.Z. and S.Q. conceived and designed the experiments; S.S. performed the experiments; C.Z. and S.S. analyzed the data; T.T. and Y.S. contributed materials; C.Z. and S.Q. wrote the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CGI	Common Gateway Interface
FCN	Fully Connected Neural Network
GRU	Gate Recurrent Unit
HTTP	Hyper Text Transfer Protocol
PHP	Hypertext Preprocessor
SGM	Sample Generation Method
SNS	Social Networking Services
SQL	Structured Query Language
SVM	Support Vector Machine
XSS	Cross-Site Scripting
URL	Uniform Resource Locator

References

1. Owasp.Org. *SQL Injection*; OWASP Foundation: Maryland, MA, USA, 2022.
2. Owasp.Org. *Cross Site Scripting (XSS) Software Attack*; OWASP Foundation: Maryland, MA, USA, 2022.
3. Bleepingcomputer.Com. *FlexBooker Discloses Data Breach, over 3.7 Million Accounts Impacted*; Bleepingcomputer: New York, NY, USA, 2022.
4. SecurityAffairs.Co. *Croatian Phone Carrier A1 Hrvatska Has Disclosed a Data Breach that Has Impacted Roughly 200,000 Customers*; SecurityAffairs.Co.: New York, NY, USA, 2022.
5. Bleepingcomputer.Com. *Toyota Halts Production after Reported Cyberattack on Supplier*; Bleepingcomputer: New York, NY, USA, 2022.
6. ItGovernance.Co.Uk. *Data Breaches and Cyber Attacks Quarterly Review: Q1 2022*; ItGovernance.Co.: London, UK, 2022.
7. Shar, L.K.; Tan, H.B.K. Defeating SQL injection. *Computer* **2012**, *46*, 69–77. [[CrossRef](#)]
8. Appiah, B.; Opoku-Mensah, E.; Qin, Z. SQL injection attack detection using fingerprints and pattern matching technique. In *Proceedings of the 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, China, 24–26 November 2017; pp. 583–587.

9. Yang, J.; Yang, P.; Jin, X.; Ma, Q. Multi-classification for malicious URL based on improved semi-supervised algorithm. In Proceedings of the 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), Guangzhou, China, 21–24 July 2017; Volume 1, pp. 143–150.
10. Ma, J.; Saul, L.K.; Savage, S.; Voelker, G.M. Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, 28 June–1 July 2009; pp. 1245–1254.
11. Pandurang, R.M.; Karia, D. A mapping-based model for preventing Cross site scripting and sql injection attacks on web application and its impact analysis. In Proceedings of the 2015 1st International Conference on Next Generation Computing Technologies (NGCT), Dehradun, India, 4–5 September 2015.
12. Xiao, Z.; Zhou, Z.; Yang, W.; Deng, C. An approach for SQL injection detection based on behavior and response analysis. In Proceedings of the 2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN), Guangzhou, China, 6–8 May 2017.
13. Gupta, M.K.; Govil, M.C.; Singh, G. Text-mining based predictive model to detect XSS vulnerable files in web applications. In Proceedings of the India Conference, Bangalore, India, 16–18 December 2016.
14. Goseva-Popstojanova, K.; Anastasovski, G.; Dimitrijević, A.; Pantev, R.; Miller, B. Characterization and classification of malicious Web traffic. *Comput. Secur.* **2014**, *42*, 92–115. [[CrossRef](#)]
15. Sahoo, D.; Liu, C.; Hoi, S.C. Malicious URL detection using machine learning: A survey. *arXiv* **2017**, arXiv:1701.07179.
16. Rathore, S.; Sharma, P.K.; Park, J.H. XSSClassifier: An Efficient XSS Attack Detection Approach Based on Machine Learning Classifier on SNSs. *J. Inf. Process. Syst.* **2017**, *13*, 1014–1028. [[CrossRef](#)]
17. Vishnu, B.A.; Jevitha, P.K. Prediction of Cross-Site Scripting Attack Using Machine Learning Algorithms. In *Proceedings of the 2014 International Conference*; ACM: Guildford, UK, 2014; pp. 1–5.
18. Gould, C.; Su, Z.; Devanbu, P.T. JDBC checker: A static analysis tool for SQL/JDBC applications. In Proceedings of the 26th International Conference on Software Engineering, (ICSE), Edinburgh, UK, 23–28 May 2004.
19. Yi, W.; Li, Z.; Guo, A. Literal Tainting Method for Preventing Code Injection Attack in Web Application. *J. Comput. Res. Dev.* **2012**, *49*, 2414–2423.
20. Wassermann, G.; Gould, C.; Su, Z.; Devanbu, P.T. Static checking of dynamically generated queries in database applications. *ACM Trans. Softw. Eng. Methodol.* **2007**, *16*, 4. [[CrossRef](#)]
21. Ma, Z.; Xue, J.H.; Leijon, A.; Tan, Z.H.; Yang, Z.; Guo, J. Decorrelation of Neutral Vector Variables: Theory and Applications. *IEEE Trans. Neural Netw. Learn. Syst.* **2017**, *29*, 129–143. [[CrossRef](#)] [[PubMed](#)]
22. Li, Q.; Wang, F.; Wang, J.; Li, W. LSTM-based SQL injection detection method for intelligent transportation system. *IEEE Trans. Veh. Technol.* **2019**, *68*, 4182–4191. [[CrossRef](#)]
23. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [[CrossRef](#)] [[PubMed](#)]
24. Stasinopoulos, A.; Ntantogian, C.; Xenakis, C. Commix: Automating evaluation and exploitation of command injection vulnerabilities in Web applications. *Int. J. Inf. Secur.* **2018**, *18*, 49–72. [[CrossRef](#)]
25. Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv* **2014**, arXiv:1406.1078.
26. Liang, J.; Zhao, W.; Ye, W. Anomaly-based web attack detection: a deep learning approach. In Proceedings of the 2017 VI International Conference on Network, Communication and Computing, Kunming, China, 8–10 December 2017; pp. 80–85.
27. Giménez, C.T.; Villegas, A.P.; Marañón, G.Á. *HTTP Data Set CSIC 2010*; Information Security Institute of CSIC (Spanish Research National Council): Madrid, Spain, 2010.
28. Zhou, C.; Sun, C.; Liu, Z.; Lau, F. A C-LSTM Neural Network for Text Classification. *Comput. Sci.* **2015**, *1*, 39–44.
29. Saxe, J.; Berlin, K. eXpose: A character-level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys. *arXiv* **2017**, arXiv:1702.08568.
30. Zhang, X.; Zhao, J.; LeCun, Y. Character-level convolutional networks for text classification. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 1–9.
31. Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks. *Andrej Karpathy. Blog* **2015**, *21*, 23.
32. Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **1994**, *5*, 157–166. [[CrossRef](#)] [[PubMed](#)]
33. Greff, K.; Srivastava, R.K.; Koutník, J.; Steunebrink, B.R.; Schmidhuber, J. LSTM: A Search Space Odyssey. *IEEE Trans. Neural Netw. Learn. Syst.* **2016**, *28*, 2222–2232. [[CrossRef](#)] [[PubMed](#)]