2021 3rd International Academic Exchange Conference on Science and Technology Innovation (IAECST) | 978-1-6654-0267-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/IAECST54258.2021.9695557

Machine Learning-based web security intrusion detection system

Chuanlin Chen[1]
[1] School of Cyberspace Security
Qilu University of Technology (Shandong Academy of Sciences)
Jinan, 250353, China

Jing Zhong[2*]
[2] School of Computer Science, School of Software, School of Cyberspace Security
Nanjing University of Information Science & Technology
Nanjing, 210044, China
[*]zhongjing1122@qq.com

Wenjie Chen[3]
[3] Financial Mathematics and Statistics
Guangdong University of Finance
Guangzhou, 510521, China

***Abstract*—In recent years, the rise of technologies such as dynamic web pages have led to the emergence of many web applications. In addition, the complexity and variety of attacks against web applications make it difficult for intrusion detection systems based on misuse detection to use constructed attack signatures for detection effectively. This paper first presents two common web attacks, SQL injection and XSS cross-site scripting attacks, their specific classification, followed by a simple optimization of the collected SQL and XSS datasets, the optimization of the model parameters using machine learning algorithms, and a comparison of the performance of the different algorithms on GPU/CPU. In addition, they were deployed in a production environment for practical testing. After experimental results and error analysis, the best algorithmic model was determined. Based on these experiments, a machine learning-based network intrusion detection system was designed without compromising the quality of the webserver.**

Keywords-Machine Learning, Intrusion Detection Systems, SVM Optimal, WEB Security

I. INTRODUCTION

As web technology develops rapidly, there are more and more security issues being faced by web applications. According to the Open Web Application Security Project (OWASP) data, SQL injection and XSS attacks are the principal vulnerabilities. SQL Injection (SQLI), where malicious code is placed in a SQL statement via web input. XSS attacks, where an attacker inserts malicious HTML code into a web page when the user is not expecting it. Security testing is an essential part of web security, assessing the security of systems and eliminating their potential danger to web applications by simulating attacks on common areas prone to security vulnerabilities. Over the past few years, many experts and academics have researched SQL injection vulnerabilities and XSS attacks and have proposed many detection and defence techniques. Meixing Le, Stavrou, and Kang[1] proposed a system in which a mapping model is implemented to detect SQL injection attacks. In this approach, static and dynamic web applications are handled. Nevertheless, this approach fails to detect XSS attacks. Pankaj Sharma [2] 2012 proposed an integrated approach to prevent SQL injection attacks & reflected cross-site scripting attacks. Parveen Sadotra [3] 2017 SQL Injection Impact on Web Server & Their Risk Mitigation Policy Implementation Techniques. After the study, we found that when detecting SQL injection and XSS attacks, the effectiveness of using different measuring methods to identify vulnerabilities varies, which affects our remediation of vulnerabilities. Therefore, we propose seven different DI-based testing algorithms to judge the measurement results by comparing accuracy and precision.

II. SQL INJECTION AND XSS CROSS-SITE SCRIPTING RESEARCH

A. SQL Injection Research

This section focuses on the principles of injection attacks and classifies injection vulnerabilities.

(1) SQL Injection Principle

It usually occurs when SQL commands are inserted into an input domain or page request web form submission or query string.

(2) SQL Injection Categories

a Tautologies: This type of injection can be used to bypass session authentication, identify injectable parameters or extract data.

b Logically Incorrect Queries: Identifies injectable parameters, identifies database fingerprints and related information or extracts data stored in the database.

c Union Query: primarily obtains sensitive data stored by the application in the background.

d Stored Procedures: Elevate privileges and execute remote malicious commands.

978-1-6654-0267-5/21/$31.00 ©2021 IEEE

e Timing Inference Query: obtains data and identifies injections.

f Second-order SQL injection: Second-order SQL injection is an excellent way to avoid filtering policies.

Statistical frequency of occurrence

```
if len(line)!=0:
    num_f=num_len/len(line)  # Digital character frequency
capital_len=len(re.compile(r'[A-Z]').findall(line))
#Calculate
all occurrences of capital letters in this line

if len(line)!=0:
    capital_f=capital_len/len(line)  #Capital letter frequency
    line=line.lower()
```

Statistical keywords

```
key_num=line.count('and%20')+line.count('or%20')
+line.count('xor%20')+line.count('sysobjects%20')
+line.count('version%20')+line.count('substr%20')
+line.count('len%20')+line.count('substring%20')
+line.count('exists%20')

key_num=key_num+line.count('mid%20')
+line.count('asc%20')+line.count('innerjoin%20')
+line.count('xp_cmdshell%20')+line.count('version%20')
+line.count('exec%20')+line.count('having%20')
+line.count('unnion%20')+line.count('order%20')
+line.count('information schema')

key_num=key_num+line. count ('load_file%20')
+line. count ('load data infile%20')+line. count ('into outfile%20')
+line. count ('into dumpfile%20')
```

B. Cross-site scripting research

This section examines and analyses the principles and classification of cross-site scripting vulnerabilities.

(1) Cross-Site Scripting Principles Analysis

An XSS vulnerability can occur whenever client software supports HTML parsing and Javascript parsing, when unintended script commands are rendered and executed by a user's browser throughout an HTML document, at the browser level of the target user of the target website.

(2) XSS categories

a Reflective XSS: The attack code for reflective XSS is present in the URL path of the user request and requires the attacker to click on a malicious link to trigger the vulnerability.

b Stored XSS: Stored XSS submissions store malicious code on the server-side. Stored XSS attacks are very stealthy and can spread XSS worms on a large scale.

c DOM XSS: DOM XSS is triggered by DOM parsing on the browser side, and DOM-type cross-site vulnerabilities are easily triggered when the client outputs HTML content directly.

III. System Architecture

This system will be divided into XSS Module and SQL Module for SQL injection and XSS cross-site scripting research. In the SQL Module, pre-process the data first and then test the seven models; in the XSS Module, classify and process the data text before testing the models.

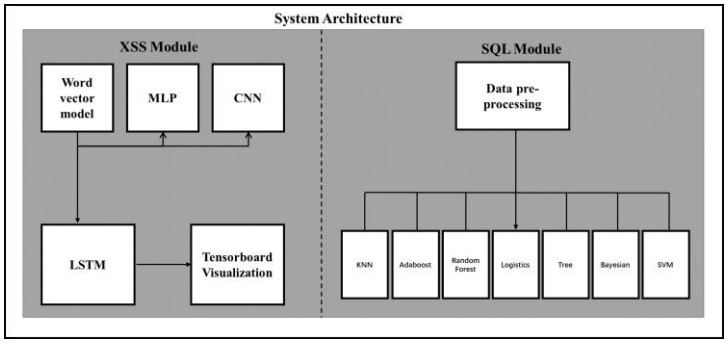

Fig. 1 System Architecture

IV. Experimentation and Analysis

A. Pre-processing of SQL statement datasets

We extracted and pre-processed (Length of each line, keywords, frequency of capital letters, frequency of numbers, percentage of spaces, percentage of unique characters, percentage of prefixes).

B. *SQL statement classification algorithm*

Seven algorithmic models were used to classify the SQL statements: SVM, Adaboost, Decision Tree, Random Forest, Logistic Stiff Regression, KNN, and Bayesian.

Randomly divide the training and test sets

```
target=arr[:,7]

clf=GaussianNB() #Create a classifier object that
clf.fit(train_data,train_target) # train the model
```

Predicting data for SQL statement classification

```
joblib.dump(clf, './file/bys.model')
print("forestrandom.model has been saved to 'file/bys.model'")
#clf = joblib.load('svm.model')
y_pred=clf.predict(test_data) #forecast
print("y_pred:%s"%y_pred)
print("test_target:%s"%test_target)
```

Verify the accuracy of the model for classifying SQL statements

```
print('Precision:%.3f' %metrics.precision_score(y_true=test_target,
y_pred=y_pred)) #Search completion rate
print('Recall:%.3f' %metrics.recall_score(y_true=test_target,
y_pred=y_pred)) #Check accuracy
print (metrics.confusion_matrix(y_true=test_target,y_pred=y_pred))
#Confusion matrix
```

C. *Performance comparison of seven SQL statement classification algorithms*

All seven models predicted the SQL statement accurately. y_pred is the result of the model prediction and test_target is the actual category.

```
y_pred:[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1.
 1. 1. 1. 1.]
test_target:[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1.]
```

Fig. 2 Predicted results

We used three parameters to analyze the accuracy of the seven models in predicting the SQL category, and we can see that all seven models [4], [5] have a perfect rate of 1.000. However, the Adaboost model has the highest accuracy rate among the seven models. Their confusion matrices only need to keep one number on the opposite diagonal as 0 and one on the positive diagonal that is not 0. Therefore, machine learning models significantly improve the efficiency and accuracy of detecting SQL injection statements.

TABLE I. PERFORMANCE COMPARISON OF SEVEN MODELS

Algorithm	Adaboost	Random Forest	KNN	Logistic
Precision	1.000	1.000	1.000	1.000
Recall	**0.962**	0.942	0.904	0.904
Confusion matrix	[[0 0] [2 50]]	[[0 0] [3 49]]	[[0 0] [5 47]]	[[0 0]] [5 47]]

TABLE II. PERFORMANCE COMPARISON OF SEVEN MODELS

Algorithm	Decision Tree	Bayesian	SVM
Precision	1.000	1.000	1.000
Recall	0.923	0.942	0.808
Confusion matrix	[[0 0] [4 48]]	[[0 0] [3 49]]	[[0 0] [10 42]]

A line graph of precision and recall for the seven models is shown in Figure 3.

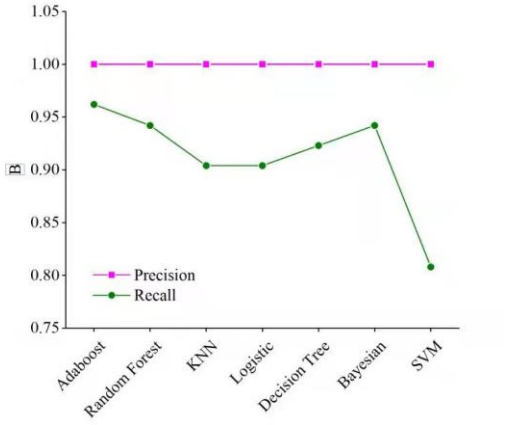

Fig. 3 Precision and recall of seven models

where precision, recall, and confusion matrix are defined as:

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

$$Recall = \frac{TP}{FN} \quad (2)$$

TABLE III. CONFUSION MATRIX

	Positive	Negative
True	True Positive (TP): Predicting positive classes to positive class numbers	True Negative (TN): Predicting negative classes to negative class numbers
False	False Positive (FP): Predicting negative classes to positive class numbers	False Negative (FN): Predicting positive classes to negative class numbers

The above TABLE I & II shows the results of comparing the detection rate and accuracy of the seven algorithm models for SQL injection. The purpose of the experiment is to select the best algorithm model for SQL statement detection, and the detection rate of each model is the same (all can thoroughly search SQL statements), among which the detection rate of the Adaboost model is the best at 0.962.

D. *Pre-processing of XSS data*

We performed a text disambiguation process on the collected samples using the text classification of the XSS data shown in the TABLE IV.

TABLE IV. TEXT CLASSIFICATION

Sample Categories	Example
Contents of double quotes	'xs'
Links	http/https
Labels	<script>
Headings	< h1>
Parameter name subject	=
Function body warning	(
Supplement composed of alphanumeric characters	Replace the number with "0" and the hyperlink with http://u

E. Data set optimization and model parameter optimization

(1) Data set optimization

Here we have data from two production environments, each with a different system, with training data taken from the higher traffic environment, selected for only six days. When deployed to the production environment, the test data source includes both environments. The purpose of reducing the amount of training data and taking data from only one environment was to examine the model's ability to generalize to the natural environment. The white sample of the original training data was only fetched from "GET" requests, and with the addition of "POST" requests, the white sample was reduced from 200,000 to 73,000, while the black sample remained the same.

(2) Optimization of model parameters

In order to reduce the resources occupied by the model, the parameters were optimized as follows.

a The dimensionality of the word vector is reduced from 128 to 32

b The number of words in a single entry is changed from "maxlen" to a fixed 200, meaning that words exceeding 200 will be truncated.

c The batch size of the neural network is reduced from 200 to 50

d The number of neurons in each model is reduced appropriately

F. Data set optimization and model parameter optimization

(1) Model algorithm performance comparison with the production environment

In order to find an effective algorithmic model for identifying XSS [6], [7], the accuracy and recall of different models in the same environment were tested, and the following TABLE V & VI was obtained. From the table, we can see that the accuracy and recall of each model is around 99%, but the SVM model has the highest detection rate and accuracy rate in the python 3.5/cpu environment.

TABLE V. PERFORMANCE COMPARISON

Algorithms	Environment	Precision	recall
SVM	Python3.5/cpu	0.998	0.99
MLP	Python3.5/gpu	0.997	0.985
Conv	Python3.5/gpu	0.998	0.996
LSTM	Python3.5/gpu	0.998	0.993
MLP	Python2.7/cpu	0.997	0.986
Conv	Python2.7/cpu	0.998	0.996
LSTM	Python2.7/cpu	0.998	0.991

TABLE VI. PERFORMANCE COMPARISON

Algorithms	Time-consuming training	Training set time consumption
SVM	43.8s	8s
MLP	85s	29s
Conv	147.3s	40s
LSTM	104.1s	51s
MLP	179.5s	76s
Conv	217s	83s
LSTM	507.6s	145s

The above TABLE V & VI are visualized as a histogram of the precision and recall of the seven models in different environments (Figure 4) and a histogram of the consuming training time of the seven models (Figure 5).

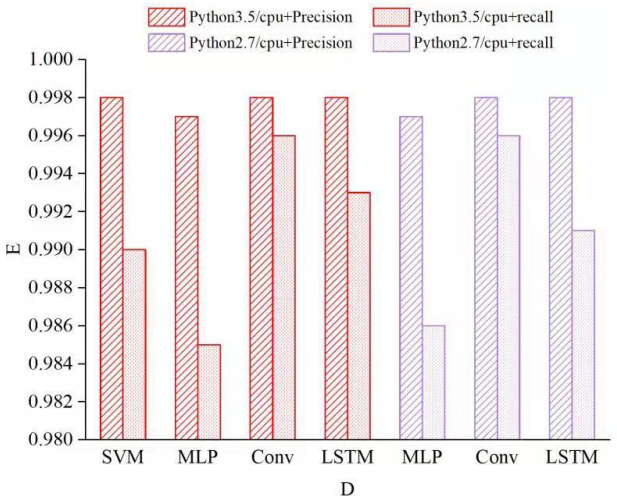

Fig. 4 Precision and recall of seven models in different environments

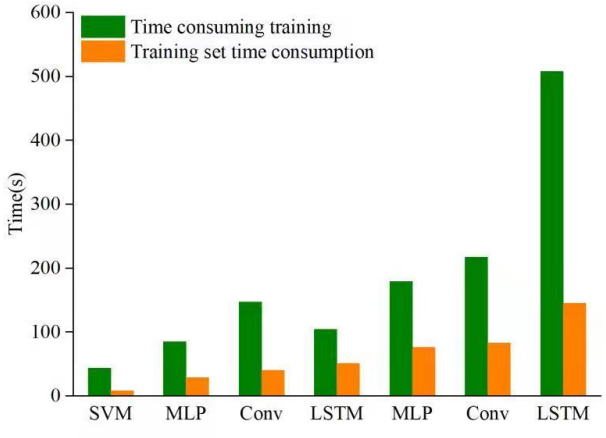

Fig. 5 Consuming training time of the seven models

Detecting data from the production environment from September 1st to October 8th, the number of alarms is shown in the TABLE VII below. The SVM generated many false alarms, running a small amount of data and then stopping (so not counted in the table), indicating that the SVM has poor generalization capabilities.

TABLE VII. COMPARISON OF PRODUCTION ENVIRONMENTS

Algorithms	Number of alarms
MLP	1002
Conv	804
LSTM	1914

The model examined data from the production environment from September 1st to October 8th, and the number of alarms is shown in the table below. The SVM generated many false positives and stopped running a small amount of data, so the table does not count. The reason for the false positives: SVM considers features independent of each other and, therefore, cannot identify word position and word-to-word relationships. For example, "< script" should have the same impact on the classification result at the second or fifth words, but SVM represents different features at different positions. The way the word list is constructed means that the black samples in the test data contain more non-UNK words than the white samples, so the black and white samples are spatially separable in the test data. However, the production environment is very complex and diverse, and the SVM lacks sufficient learning capability when it comes to average data with large numbers of non-UNK words.

G. *Deep learning for false alarm analysis*

The false positives generated by the three algorithms were analyzed manually, and none of them was XSS. The false positives generated can be divided into two categories: standard data is identified as XSS, contains referrer parameters or URLs, login behaviour, post data as HTML or Javascript. The other type of false alarm is exciting, as the neural network identifies other types of attacks as XSS, such as SQL injection, various n-day (strust2, phpcms), and web shell.

H. *Target machine testing*

After more than a month of testing the actual data without detecting XSS, a target machine was built to simulate the attack traffic and examine the detection capability. The target machine is DVWA, which includes a reflection type and a storage type vulnerability. In addition to the exploit poses for each difficulty level, some complex XSS payloads were selected from the internet.

V. EXPERIMENTATION CONCLUSION

This paper proposes seven algorithms for measuring vulnerabilities based on artificial intelligence and judging the results by comparing performance such as precision and recall. The experimental results show that the system effectively detects SQL injection and XSS attacks, and the following conclusions are drawn.

(1) Deep learning has a better generalization capability than traditional machine learning and can learn features of higher dimensionality.

(2) LSTM has better detection capability but at the expense of accuracy.

(3) As the model is a binary model, when faced with other types of anomalous data, it considers them closer to XSS. To reduce such false positives can add other types of attack data, train a multi-classification model, or train a "normal-anomaly" binary model.

(4) Deep learning models have more parameters and require more training data, and the amount of data used in this paper is not sufficient for deep learning.

REFERENCES

[1] Hiteshkumar, C., Narendra, B., & Sushil, S. (2015). DoubleGuard: Detecting Intrusions in Multi-tier Web Applications. International Journal of Advanced Research in Computer and Communication Engineering, 473-476.

[2] Sharma, P., Johari, R., & Sarma, S.S. (2012). Integrated approach to prevent SQL injection attack and reflected cross site scripting attack. International Journal of System Assurance Engineering and Management, 3, 343-351.

[3] Sadotra, P., & Sharma, C. (2017). SQL Injection Impact on Web Server and Their Risk Mitigation Policy Implementation Techniques: An Ultimate solution to Prevent Computer Network from Illegal Intrusion. International Journal of Advanced Research in Computer Science, 8, 678-686.

[4] R, Jothi K et al. "An Efficient SQL Injection Detection System Using Deep Learning". 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE) (2021):442-445.

[5] Joshi, Anamika and V. Geetha. "SQL Injection detection using machine learning". 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT) (2014): 1111-1115.

[6] Lei, Li et al. "XSS Detection Technology Based on LSTM-Attention". 2020 5th International Conference on Control, Robotics and Cybernetics (CRC) (2020): 175-180.

[7] Habibi, Gulit and Nico Surantha. "XSS Attack Detection With Machine Learning and n-Gram Methods". 2020 International Conference on Information Management and Technology (ICIMTech) (2020): 516-520