Article

Resolving Cross-Site Scripting Attacks through Fusion Verification and Machine Learning

Jiazhong Lu †, Zhitan Wei †, Zhi Qin, Yan Chang and Shibin Zhang *

School of Cybersecurity, Chengdu University of Information Technology, Chengdu 610225, China
* Correspondence: cuitzsb@cuit.edu.cn
† These authors contributed equally to this work as co-first authors.

Abstract: The frequent variations of XSS (cross-site scripting) payloads make static and dynamic analysis difficult to detect effectively. In this paper, we proposed a fusion verification method that combines traffic detection with XSS payload detection, using machine learning to detect XSS attacks. In addition, we also proposed seven new payload features to improve detection efficiency. In order to verify the effectiveness of our method, we simulated and tested 20 public CVE (Common Vulnerabilities and Exposures) XSS attacks. The experimental results show that our proposed method has better accuracy than the single traffic detection model. Among them, the recall rate increased by an average of 48%, the F1 score increased by an average of 27.94%, the accuracy rate increased by 9.29%, and the accuracy rate increased by 3.81%. Moreover, the seven new features proposed in this paper account for 34.12% of the total contribution rate of the classifier.

Keywords: XSS attack; traffic detection; payloads; fusion verification

MSC: 68T09

check for updates

Citation: Lu, J.; Wei, Z.; Qin, Z.; Chang, Y.; Zhang, S. Resolving Cross-Site Scripting Attacks through Fusion Verification and Machine Learning. *Mathematics* **2022**, *10*, 3787. https://doi.org/10.3390/math10203787

Academic Editors: Jianping Gou, Weihua Ou, Shaoning Zeng and Lan Du

Received: 9 September 2022
Accepted: 8 October 2022
Published: 14 October 2022

1. Introduction

XSS (cross-site scripting) attacks have caused enormous damage to economics and individual privacy [1]. Moreover, XSS attacks have been adjusted from the seventh to the third most common in the newly released 2021 version of OWASP (Open Web Application Security Project) Top 10 [2].

Normally, there are three types of XSS attack, namely reflected XSS attack, stored XSS attack, and DOM-based XSS attack. These three attack types usually use the GET or POST methods of the HTTP protocol to inject malicious code at the URL or POST Body. Reflected XSS usually injects malicious code into the URL, which can only be triggered in the current browser and does not store malicious code permanently. The malicious code of stored XSS is injected into the server-side database through vulnerabilities, which can cause long-term information leakage and other hazards. In fact, we can think of DOM-based XSS as a special kind of reflected XSS. Its malicious code can only be triggered in the current browser when it runs the script on the client side for front-end page rendering.

In general, there are two popular methods to defend against XSS attack: static analysis and dynamic analysis. Static analysis finds vulnerabilities by scanning the source code to analyze information such as lexical, grammar, control flow, data flow, and other information. It is in the development and coding phase of the program that requires developers to master a lot of security-related knowledge. Dynamic analysis inputs test data during program execution and analyze the output information to determine whether there are loopholes. However, this method relies on the completeness of the test data.

In the face of frequent variations in XSS payloads, it is hard for traditional XSS detection to have a pleasing result. There are some factors that have a significant impact on the results. For example, traditional XSS detection requires a large number of manual participation and the integrity of the attack vector.

Recently, machine learning techniques have been widely used in XSS attack detection and achieved good results. However, most of the detection approaches based on machine learning only focus on one of the traffic or XSS payloads. On the one hand, traffic detection has certain timeliness, but it is difficult to accurately detect and identify XSS attacks. On the other hand, XSS payload detection has a certain degree of accuracy, but it lacks timeliness. Another reason for this may be that there is currently no public dataset that includes both normal traffic and XSS attack traffic (the only type of attack in the attack traffic is XSS).

As a result, a lot of XSS detection methods cannot meet the dual requirements of timeliness and accuracy in real environments, and the pros and cons of a single model will directly affect the performance of the entire detection model. This leads to the problem of low accuracy and a high false negative rate for a single model.

The primary contribution of this paper is to propose a fusion verification method that combines traffic detection and XSS payload detection. Previously, both traffic detection and XSS payload detection have been separately applied to XSS detection. However, to the best of our knowledge, fusion verification methods combining the two methods have not been reported in the literature for detecting XSS attacks. The main contributions of this paper are:

- Combine traffic detection and XSS payload detection for XSS attack detection through fusion verification
- Propose seven new payload detection features through feature extraction based on XSS attack methods
- Obtain datasets of normal traffic and XSS attack traffic by simulating public CVE

2. Related Work

For the study of XSS attack detection, network security researchers have successively put forward some effective detection methods and preventive measures.

In terms of static analysis, Medeiros et al. [3] proposed a cross-site scripting vulnerability detection method combining static source code analysis and data mining in 2015. The accuracy of XSS vulnerability detection and the effect of fixing code as improved by this method, but the disadvantage was false positives. Choi et al. [4] proposed an HXD (Hybrid XSS Detection) system. The system used both static string analysis and dynamic browser rendering with a black-box detection approach. Experimental results showed that HXD had a low false positive rate. Mohammadiet al. [5] detected XSS vulnerabilities through an automatic unit testing method. They preferred to automatically construct an XSS vulnerability unit test from each web page; the test input pair framework was then automatically generated using a grammar-based attack generator, which was then evaluated. The proposed method reduced the error rate of XSS vulnerabilities. In 2019, YAN et al. [6] proposed a PHP code vulnerability detection method based on sensitive path and taint analysis. The method first converted the background code of the web application into the intermediate representation of the code, such as the abstract syntax tree, then found the slot (dangerous function), then determined the sensitive path through the slot, and finally performed taint analysis on this path to determine whether the vulnerability exists. However, the disadvantages of static analysis were obvious, it relied on a lot of manual work by human experts with knowledge of both programming and security domains, and the source code was usually not open-source.

In terms of dynamic analysis, Parameshwaran et al. [7] designed a DOM-based XSS test platform, which was based on taint analysis in 2015. The platform included a vulnerability generator and a detection engine. Experiments showed that the method had an excellent effect on detecting DOM-based XSS attacks. Wang et al. [8] proposed a TT-XSS framework to detect DOM-based XSS using dynamic taint analysis. The application dynamically analyzed the collected URLs that were then sent to the taint tracking analysis module, the obtained taint trajectories were sent to the automatic vulnerability verification module, and the verification module was completed by generating attack vectors from taint trajectories. In 2021, Khalaf et al. [9] proposed an algorithm that allowed attack detection and prevention using an input validation mechanism. This approach supported web security testing by providing an easy-to-use and accurate vulnerability prediction model and validation method, which had the advantage of having a very low false positive rate. However, this method relied on the completeness of the testing dataset. If the testing dataset was not perfect or faced deformation attacks, it would produce a high false negative rate. In addition, this is a common problem for all dynamic analyses.

In recent years, zero-day attacks and deformation attacks are common, and it is difficult for traditional static analysis and dynamic analysis to play an effective role in XSS detection. Therefore, a large number of scholars have introduced machine learning technology for XSS detection and achieved good results. Zuhair et al. [10] also extracted features from Web pages and URLs but made a mixed feature subset division, combined with phishing attacks, and finally used the SVM algorithm for training and testing. Rathore et al. [11] proposed a machine learning method based on URLs, web pages, and SNSs to detect XSS attacks in 2017, extracted twenty-five XSS attack features, and used ten classifiers for detection. To achieve better performance, Hosseini et al. [12] proposed a model for detecting malicious crawler behavior using machine learning techniques and tested and compared several machine learning algorithms, such as Bayesian networks, SVM, and decision trees. Finally, in this experiment, it was found that the SVM-based model had higher detection accuracy for malicious crawlers and extracting effective features could improve the detection accuracy. In 2021, Hu et al. [13] designed and implemented an XSS attack detection model for web applications. This model added the verification code recognition function to solve the problem of submitting data to the server just by entering the verification code; this model had a low false positive rate. Malviya et al. [14] developed a web browser for machine learning classification to mitigate XSS attacks. Experimental results showed that the proposed method outperforms other proposed methods in classification accuracy, recall, precision, and F1-score. Mokbal et al. [15] proposed a novel XSS attack detection framework based on the ensemble learning technique for web applications, which used the XG boost (Extreme Gradient Boosting) algorithm and the extreme parameter optimization method. The proposed framework passed multiple tests on the testing dataset, and the accuracy could reach 99.59%. Soltani et al. [16] proposed a framework for a DID (Deep Intrusion Detection) system. The authors deployed and evaluated offline IDS (Intrusion Detection System) following this framework. Experiments showed that the evaluation indicators, such as the precision rate and recall rate, of this method, reached 0.992 and 0.998, respectively. In addition, the shortage of high-quality data has always been a key problem in machine learning. Multi-fidelity classification algorithms [17–19] solve this type of problem by incorporating information from other sources that can be obtained at a low cost while maintaining good correlation. In this regard, it can also be applied to the XSS attack detection model in the future to improve the generalization ability of the model.

Our previous work [20] can detect XSS attacks more accurately by using machine learning to jointly detect traffic and logs and at the same time, trace the process of XSS attacks in the entire network, but it needs to collect a large number of network device logs for analysis.

To sum up, the current XSS attack detection approaches still have the following problems:

- XSS remains one of the most serious and common types of attacks. Therefore, a more effective detection method is needed to defend against XSS attacks.
- The pros and cons of a single model of the existing detection methods will directly affect the effectiveness of the entire detection model.
- Existing detection methods have a high false negative rate in the face of the frequent variations in XSS payloads, which needs to be reduced.
- There is currently no public dataset that includes both normal traffic and XSS attack traffic (the only type of attack in the attack traffic is XSS).

Therefore, this paper focuses on developing a fusion verification method. We obtain a real-world experimental dataset by simulating XSS vulnerabilities in CVE (Common Vulnerabilities and Exposures) and capturing network traffic on the web server side. Then we combined traffic detection with XSS payload detection to form a fusion verification method to defend against XSS attacks. Moreover, this method combines the timeliness advantages of traffic detection and the accuracy advantages of payload detection. We expect that this method can improve the performance of detection models and solve the problems that existing solutions have that make it difficult to meet actual needs.

3. Proposed Methodology

Figure 1 shows the overall framework for detecting the XSS proposed in this paper. First, the original dataset of the experiment is obtained by reproducing the CVE vulnerability. Then we use the rdpcap function of the Scapy library in Python to read the pcap file of the original dataset and summarize the data according to the upstream and downstream of the two-way communication. In addition, it is divided into two detection modules, one is traffic detection and the other is XSS payload detection. We extract the traffic dataset and the payload dataset separately through different modules. The two modules perform preprocessing and feature extraction, respectively, to form a data format that can be recognized by machine learning input. Next, the two modules separately perform preprocessing and feature extraction to form a recognizable data format for machine learning input and send it to the classifier for detection. Due to the particularity of the traffic itself, we found that each flow in the pcap packet corresponds to multiple payloads at the same time, and each result of the traffic detection module may correspond to the results of multiple payload detection modules. Therefore, we can combine the results of the two modules by matching the source port feature (src_port) common to both detection modules. Finally, the final detection result is obtained through the fusion verification of the two detection models so as to improve the detection performance of the entire model.

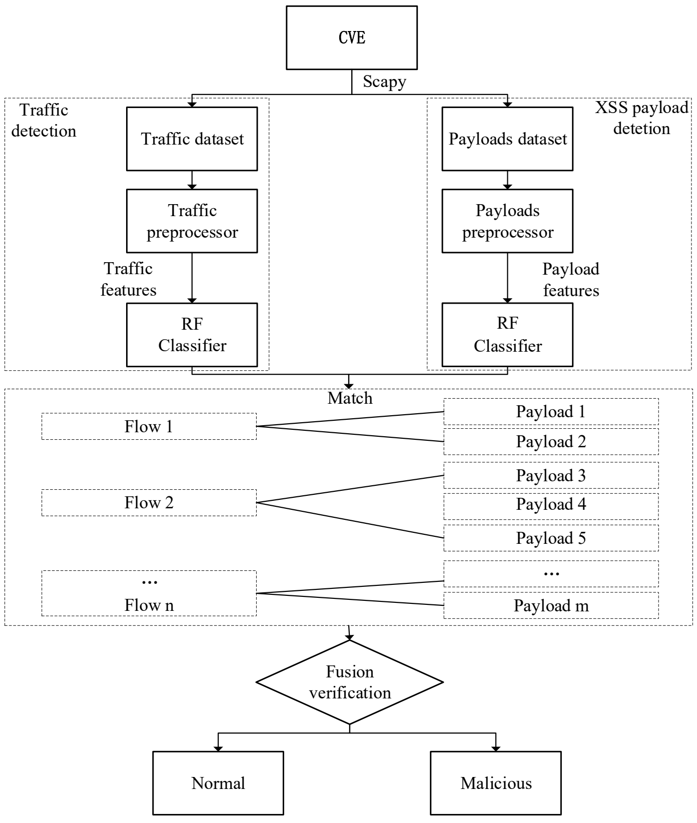


Figure 1. The framework of the proposed method.

3.1. CVE Vulnerability Set

This paper targets the widely used content management system—WordPress [21] (43.0% of websites worldwide use WordPress). From NVD [22] (National Vulnerability

Database), we have selected 10 recent XSS vulnerabilities for both reflected XSS and stored XSS. The specific CVE list is shown in Table 1. Then, the original dataset is formed by simulating locally and using WireShark [23] to capture the traffic packets of the reproduced process, and the dataset format is pcap packet.

Table 1. CVE list.

XSS Type	Vulnerability Number
Reflected XSS	CVE-2021-25067, CVE-2021-24234, CVE-2021-24180, CVE-2021-24225, CVE-2021-24436, CVE-2021-24437, CVE-2021-24452, CVE-2021-25041, CVE-2021-25047, CVE-2021-25065
Stored XSS	CVE-2021-25046, CVE-2021-24988, CVE-2021-24315, CVE-2021-24528, CVE-2021-24658, CVE-2021-24518, CVE-2021-24505, CVE-2021-24504, CVE-2022-1915, CVE-2022-1896

3.2. Traffic Features Extraction

After detection and analysis, it is found that the XSS attack traffic is different from the normal traffic. Since XSS attack traffic not only needs to load normal web pages but also needs to load malicious js files or external malicious links, resulting in extra network resources and system resources. Thus, the packets of XSS attack traffic are generally larger.

A total of 1947 flows have been extracted for analysis in this paper. Two types of features have been used for learning: traffic-related features and time-related features, which help the classifier to distinguish between normal traffic and XSS attack traffic. Moreover, the traffic-related features include the five-tuple features of the communication process (due to the particularity of the format of the IP address itself, the source IP address and the destination IP address are omitted), as shown in Table 2. This experiment has used enough traffic to reflect the real network environment and real traffic features.

Table 2. Traffic features and descriptions.

Feature Type	Feature Name	Descriptions
Traffic-related features	proto	Transfer protocol number
	src_port	Source port
	dst_port	Destination port
	up_pkts	Total number of upstream packets
	dw_pkts	Total number of downlink packets
	up_pl_bytes	Total uplink load
	dw_pl_bytes	Total downlink load
	up_min_plsize	Upstream minimum payload
	dw_min_plsize	Downlink minimum payload
	up_avg_plsize	Upstream load average
	dw_avg_plsize	Downstream load average
	up_max_plsize	Upstream maximum payload
	dw_max_plsize	Downlink maximum payload
	up_stdev_plsize	Upstream load variance
	dw_stdev_plsize	Downlink load variance
Time-related features	duration	Stream duration
	up_avg_ipt	Average time interval of upstream packets
	dw_avg_ipt	Average time interval of downlink packets
	up_min_ipt	Uplink minimum time interval
	dw_min_ipt	Downlink minimum time interval
	up_max_ipt	Uplink maximum time interval
	dw_max_ipt	Downlink maximum time interval
	up_stdev_ipt	Upstream time interval variance
	dw_stdev_ipt	Downlink time interval variance

3.3. Payload Features Extraction

In this section, after an in-depth study of XSS attack methods and causes, we have summarized three representative attack methods from the attackers' point of view. Then we extracted seven attribute features of the payloads according to the summarized attack methods.

3.3.1. XSS Attack Methods

(1) Script: Script injection can be divided into static script injection and dynamic script injection. Static script injection usually constructs malicious code within <script></script> tags to trigger scripts. Dynamic script injection refers to triggering the browser to introduce external malicious links through the src attribute of the script tag.

(2) JavaScript pseudo-protocol: XSS attack using JavaScript pseudo-protocol is also a common injection method. The JavaScript pseudo-protocol treats the segment after the code "javascript:" as a JavaScript script and executes it.

(3) Inline events: JavaScript interacts with HTML through DOM events. The HTML DOM allows JavaScript to react to HTML events and execute JavaScript when events occur. XSS attack can use DOM events to bind malicious code. Most DOM events have names starting with "on", and most HTML tags can use the on-event to trigger script code. Table 3 shows some on-events.

Table 3. Some on-events and descriptions.

Attribute	Descriptions
onerror	Run the script when an error occurs
onload	Run the script when the document loads
onfocus	Run script when window gets focus
onclick	Run a script when the mouse is clicked
onmouseover	Run a script when the mouse pointer moves over an element

Table 4 shows examples of three XSS attack methods:

Table 4. XSS attack examples.

Methods	Examples
Script	<script>alert(123);</script>
JavaScript pseudo-protocol	<iframe src="javascript:alert('xss')">
Inline events	<img src=# onerror="alert(document.cookie)">

3.3.2. Attribute Features

Usually, experienced attackers will change the encoding or capitalization of malicious code to carry out deformation attacks. Therefore, this paper has preprocessed the extracted sentences to convert them into original sentences. The preprocessing includes lowercase conversion, URL decoding, HTML decoding, JavaScript decoding, ASCII decoding, Unicode decoding, and URL decoding twice. Values are then extracted from the processed dataset to fit the features proposed in this paper.

Through extensive research on XSS attack methods and analysis of their lexical features, we have found that text characters commonly found in malicious code are often combined with certain fixed symbols. Therefore, matching the combined form can reduce the detection of false positive rate compared to just matching text characters. The following seven attribute features are summarized:

(1) HTML_Tags

HTML tags in XSS attacks typically appear more frequently than text loads in normal traffic. In HTML tags, the label starts with a left angle bracket. For example, <script, <iframe, and <img in Table 5 appear in the form of left angle brackets plus script, iframe, and img characters. Therefore, the combination of the left angle bracket and the label character is classified into a class of features.

Table 5. Seven new attribute features.

Features	Examples
HTML_Tags	<script, <img, <body, etc.
JavaScript	javascript:
On_Event	onerror=, onmouseover=, onload=, etc.
Function_Body	alert(, confirm(, eval(, etc.
Document_Object	document.cookie, etc.
Third_Party_Links	src=, href=, http:, https:, //
Delimiter	space,/, +

(2) JavaScript

The JavaScript pseudo-protocol is usually combined with HTML tags to form malicious code, such as <iframe src="javascript:alert('xss')">, where the code feature that will always appear is "javascript:".

(3) On_Event

HTML5 allows browsers to trigger scripts through various events. For example, in the malicious code "<img src=#onerror="alert(document.cookie)">", the attacker deliberately sets the src attribute of the img tag to be wrong and then uses the onerror event (run the script when an error occurs) to trigger the malicious script. Therefore, the alert function is triggered here, causing the cookie to be leaked. The features of the event attribute are the form of the on-event followed by an equal sign, such as "onerror=".

(4) Function_Body

Attackers can use some "dangerous functions" in JavaScript to steal sensitive information. For example, the "alert()" function is often used to pop up a dialog box. If an attacker combines it with the document object, the purpose of stealing cookies can be achieved. The code feature of the JavaScript function body is "alert()", which is obviously different from ordinary characters.

(5) Document_Object

The document object is the root node of the HTML document. An attacker can use the "document.write" property to write JavaScript code to the document or use "document.cookie" to return all cookies associated with the current document. Its code feature is "document."

(6) Third_Party_Links

In order to better conceal cross-site scripting attacks, experienced attackers will build an XSS attack server to receive and store the stolen sensitive information. As a result, there will be third-party links in the attack traffic, which are mostly characterized by a combination of src or href and third-party links.

(7) Delimiter

Delimiters, such as spaces, are unavoidably used within HTML tags due to the grammatical nature of HTML. Therefore, attackers must use delimiters to construct attack statements when exploiting cross-site scripting vulnerabilities. "space", "/", and"+" are known to be used as delimiters for malicious code.

In this paper, we have added 7 new features to the 30 features extracted by Zhou and Wang (2019) [1], totaling 37 attribute features of the XSS payloads. Table 5 shows the seven new attribute features added in this paper:

3.4. Fusion Verification

Both the traffic detection module and the XSS payload detection module can present the malicious or normal status of the current stream or payload in binary form. In this paper, we have adopted the fusion verification method. If either of the two detection modules declares that the current detection sample is malicious, it is considered to be malicious. In addition, if both of them declare that it is normal, it is considered to be normal.

In this paper, Boolean variables F_v and P_v are used to represent the detection results of the traffic detection module and the detection results of the XSS payload detection module, respectively. The Boolean variable R_s is used to represent the final result of fusion verification, and its calculation formula is as follows:

$$Rs = Fv \vee Pv \quad (1)$$

It is easy to know from Formula (1) that there are four cases in total. In these four cases, the final result is normal only when the traffic detection determines that it is normal and the payload detection determines that it is normal. In other cases, the result is judged to be malicious.

3.5. Random Forest

The research of a large number of scholars shows that the ensemble method has good performance in classification performance and robustness in the face of overfitting. Therefore, these kinds of algorithms are very popular in the field of machine learning. In this paper, the random forest algorithm has been used as the classification technique of the experiment. Random forest is an ensemble algorithm based on decision tree, which not only has good scalability but is also easy to use. The principle of random forest is to build a strong model with better generalization performance and less overfitting by separately averaging multiple decision trees affected by large variance.

In this paper, the random forest algorithm has been used as the classifier. The random forest algorithm does not need to worry about the choice of hyperparameter values, and pruning it is usually not necessary because of its strong resistance to noise from a single decision tree. In this experiment, we have taken the size of the training dataset as the size, n, of the bootstrap samples in order to obtain a better bias-variance tradeoff. We set the number of features, d, in each split to a value less than the total number of features in the training dataset. We have used the random forest classifier already implemented by scikit-learn with relatively reasonable parameter settings. The default value is d $= \sqrt{m}$, where m represents the total number of features in the training dataset. Additionally, we have chosen entropy as the criterion used for splitting nodes. We have set the value of the n_estimators parameter of the number of decision trees to 100. Because when the n_estimators parameter reaches 100, the accuracy of the model no longer increases. We have set the number of parallel computations, n_jobs, to 10 to use the multi-core computer parallel computing model.

4. Experiments and Discussions

4.1. Experimental Dataset

This paper has formed a traffic dataset containing normal traffic and XSS attack traffic by simulating the CVE. This dataset is called “CVE traffic”. “CVE traffic” contains 1747 normal traffic and 200 XSS attack traffic. Then we used Scapy’s rdpcap function to extract the XSS payload dataset, referred to as “CVE payloads”. It contains 10083 normal records and 231 XSS payloads.

XSS payloads [24] have been collected from GitHub and used as a training dataset with a total of 151,658 records, including 135,507 normal records and 16,151 XSS payloads. The testing dataset has been extracted from the traffic dataset above through the rdpcap function of the Scapy library.

The specific information on the experimental datasets is shown in Table 6

Table 6. Experimental dataset.

Datasets	Normal	XSS
CVE traffic	1747	200
CVE payloads	10,083	231
XSS payloads [24]	135,507	16,151

4.2. Experimental Results

This experiment uses twenty-fold cross-validations to assess the performance of the model. In this method, 19 of the 20 CVE traffic datasets are used as training datasets, and the remaining one is used as the test dataset. Additionally, each of the 20 subsets is only used once as a test dataset. The cross-validation process has been repeated 20 times, and the average of the twenty results for each CVE are taken as the result of this experiment. Then, we used the fusion verification method mentioned in Section 3.4 of this paper to take the average of 20 results for each CVE traffic detection result and XSS payload detection result as the final result of our method.

This experiment aimed to solve a typical binary classification problem. As shown in Table 7, we use a confusion matrix to represent the results.

Table 7. Confusion matrix.

	Actual XSS	Actual Normal
Predicted XSS	TP	FP
Predicted Normal	FN	TN

The confusion matrix is divided into four categories. TP (True Positive) means the number of correctly classified as attack samples, and FP (False Positive) means the number of normal samples classified as attack samples. In addition, TN (True Negative) means the number of correctly classified as normal samples, and FN (False Negative) means the number of attack samples classified as normal samples. This paper evaluates the accuracy, precision, recall, and F1 score of the experimental results. The calculation formulae are as follows:

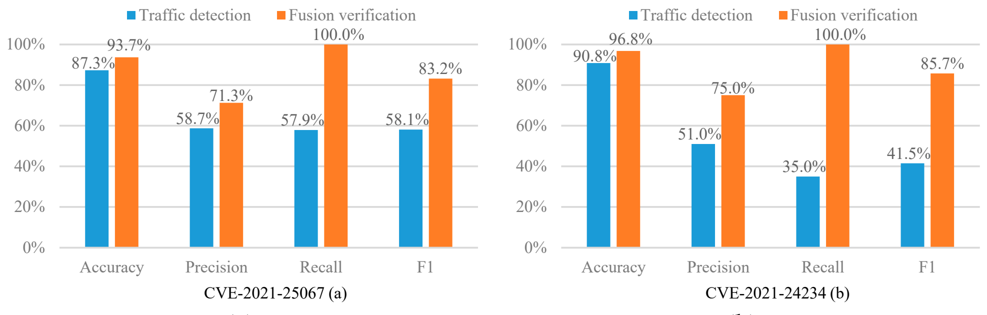
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

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

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

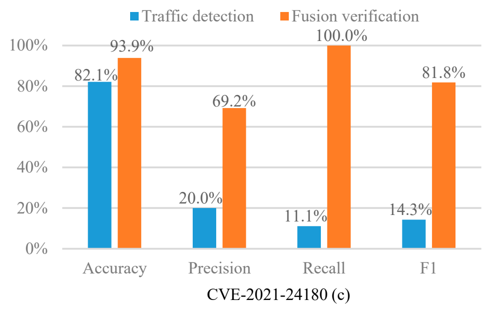
$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{5}$$

The experimental results are shown in Figure 2.

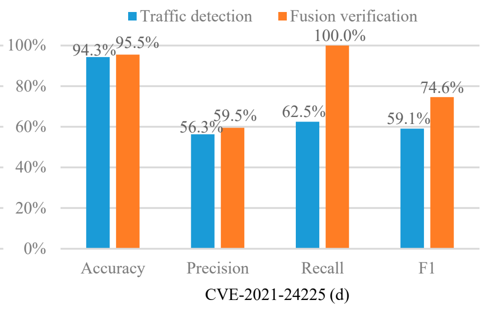


(**a**) (**b**)

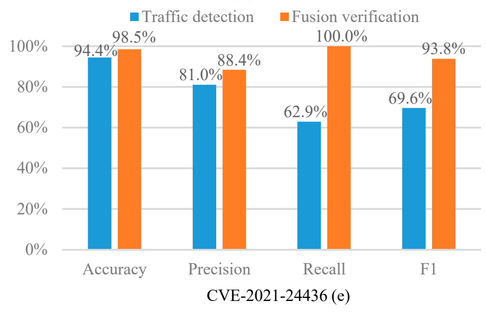
Figure 2. *Cont.*



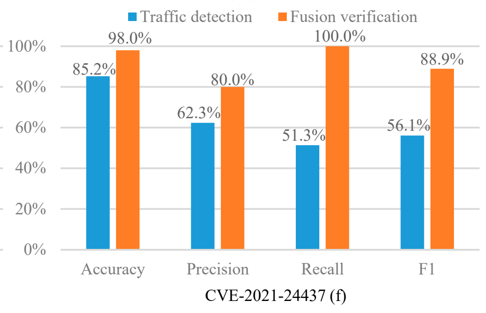
(c)



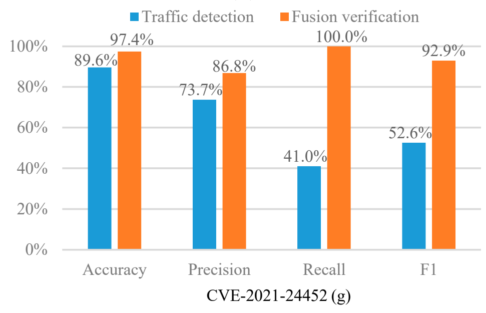
(d)



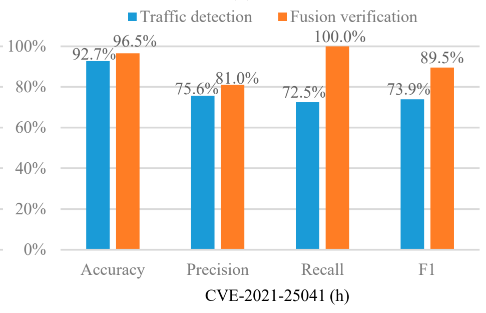
(e)



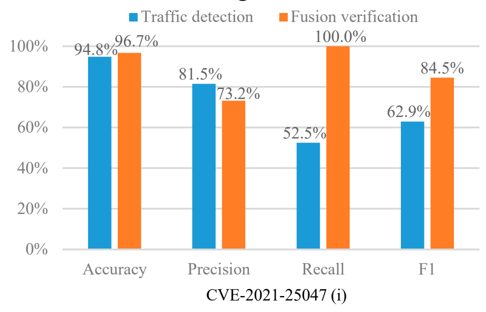
(f)



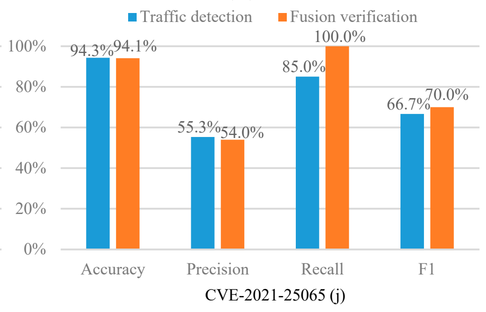
(g)



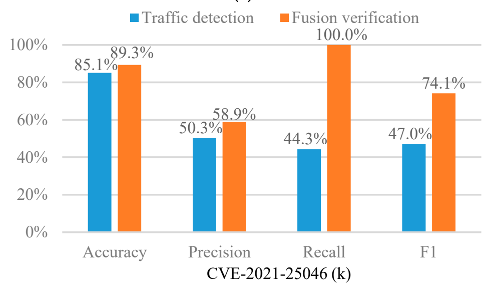
(h)



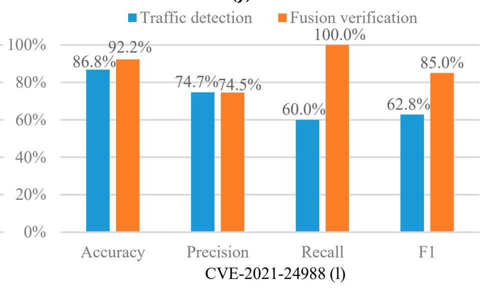
(i)



(j)



(k)



(l)

Figure 2. *Cont.*

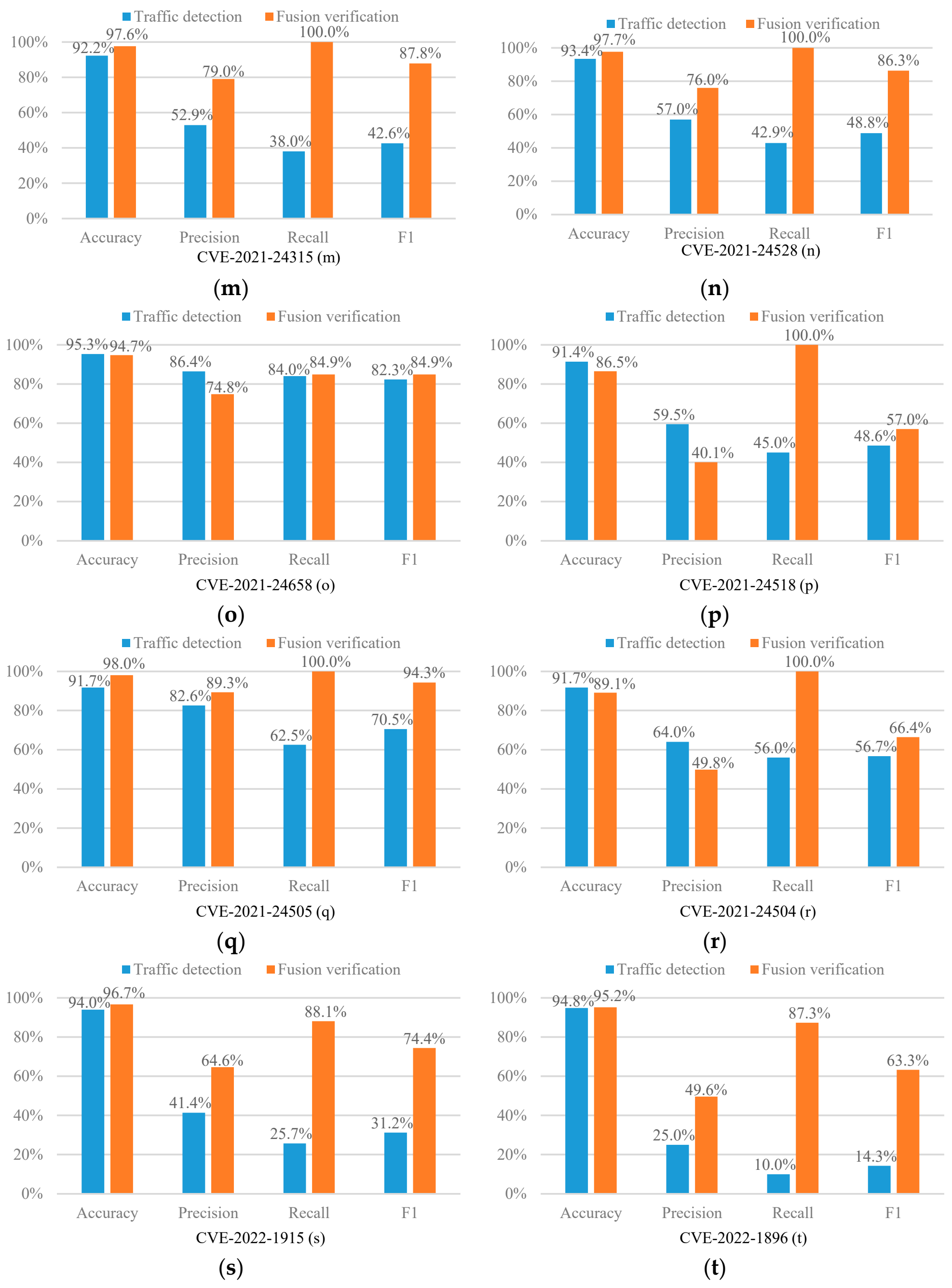


Figure 2. Experimental results (**a**–**t**).

As can be seen from Figure 2, in 17 out of 20 CVE experiments, the recall of our fusion verification method can reach an astonishing 100%. We can know from Figure 3 that under such a high recall rate, our accuracy is an average of 94.9%, which also remains at a high level. Therefore, the fusion verification method can effectively defend against XSS attacks. In addition, as shown in Figure 3, the average accuracy, precision, recall, and F1 score of this method are significantly improved compared to the single traffic detection model. Among them, the average improvement in the recall rate is as high as 48%, the average increase in F1 score is as high as 27.94%, the average increase in precision is 9.29%, and the average increase in accuracy rate is 3.81%. The results show that our proposed fusion verification model outperforms the single traffic detection model. However, the number of experimental samples in the load detection process is relatively small. Therefore, the performance of a few fusion validation models is slightly lower than that of a single detection model. In this regard, we consider using multi-fidelity classification algorithms in future research and experiments to solve the problem caused by fewer training samples.

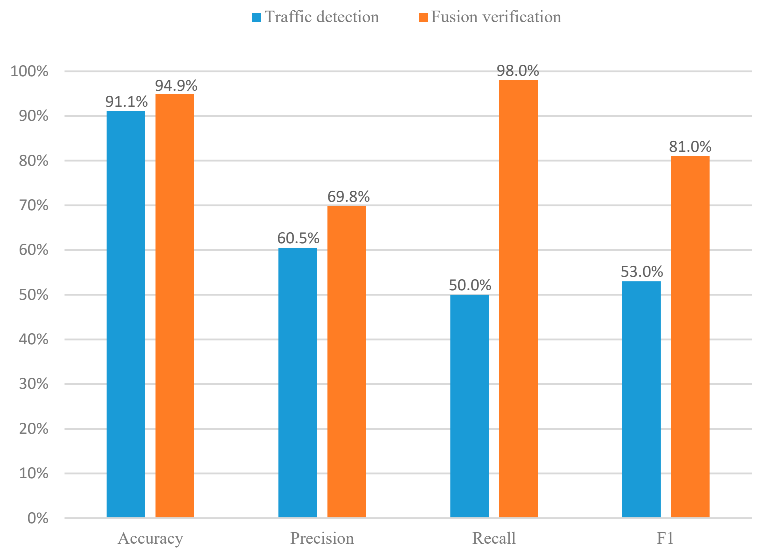


Figure 3. Average performance comparison.

In addition, taking XSS payloads [24] as the dataset, with a ratio of 7:3 between the training set and test set, random forest is used to evaluate the importance of 37 features of the payload detection link used in this paper. Among them, there are 30 features whose contribution rate is larger than 0.01%, as shown in Figure 4. The first is the feature "Function_Body" proposed in this paper, whose contribution rate is as high as 23.95%. Moreover, the total contribution rate of the seven features in this paper is as high as 34.12%. This means that it is feasible to extract detection features by summarizing XSS attack methods in this paper, and it has better generalization and can detect variations in attacks more effectively.

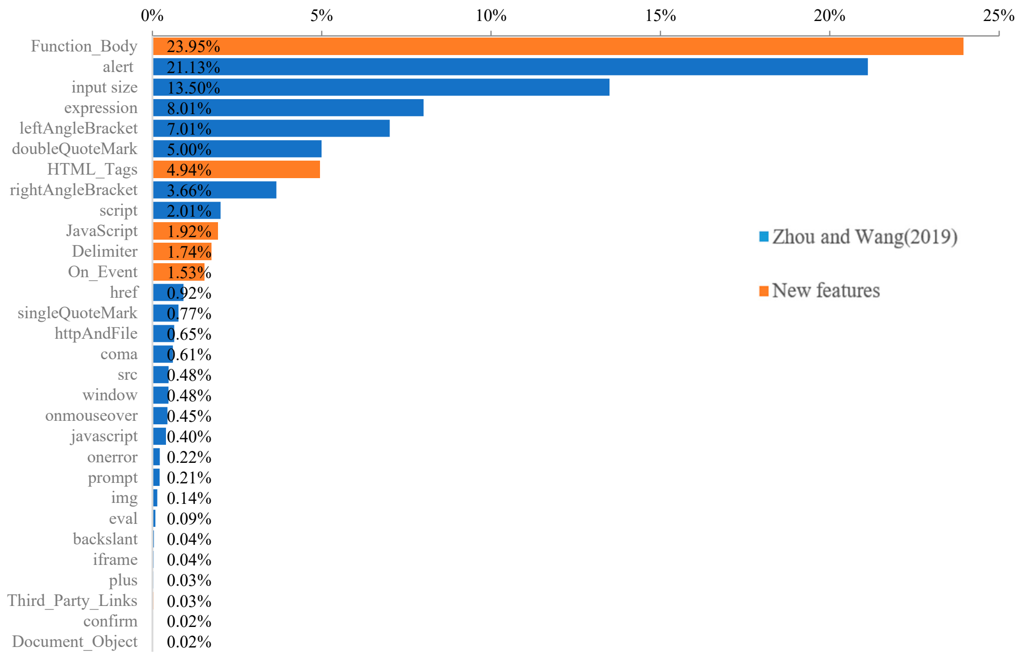


Figure 4. Assess the importance of the features of Zhou and Wang (2019) [1] and the newly proposed features in this paper.

5. Conclusions

We propose a fusion verification method that combines traffic detection with XSS payload detection to effectively detect XSS attacks. The results show that the method proposed in this paper has significant advantages for reducing the false negative rate of the model. Under the premise of uniform sample distribution, there will be almost no false negatives. Therefore, the fusion verification method can effectively defend against XSS attacks. Moreover, compared with the traditional single-flow detection model, the average recall rate of this method, F1 score, precision, and accuracy rate is increased by 48%, 27.94%, 9.29%, and 3.81%, respectively. Further, the seven new features of the XSS payloads proposed in this paper account for 34.12% of the total contribution rate of the 37 features.

However, the method proposed in this paper has certain limitations. The cost of keeping the false negative rate low is that the false positive rate of the entire model will increase. In the follow-up research, we will try to solve the existing problem.

Author Contributions: Data curation, Z.Q.; Resources, Y.C.; Software, Z.W.; Writing—original draft, S.Z.; Writing—review & editing, J.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (Grant No. 62102049). Secondly, thanks to the Key Research and Development Project of Sichuan Province (No. 2022YFS0571, No. 2021YFSY0012, No. 2020YFG0307).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could appear to influence the work reported in this paper.

References

1. Zhou, Y.; Wang, P. An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence. *Comput. Secur.* **2019**, *82*, 261–269. [CrossRef]
2. Open Web Application Security Project. OWASP Top Ten. Available online: https://owasp.org/www-project-top-ten/ (accessed on 25 September 2022).
3. Medeiros, I.; Neves, N.; Correia, M. Detecting and removing web application vulnerabilities with static analysis and data mining. *IEEE Trans. Reliab.* **2015**, *65*, 54–69. [CrossRef]
4. Choi, H.; Hong, S.; Cho, S.; Kim, Y.-G. HXD: Hybrid XSS detection by using a headless browser. In Proceedings of the 2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT), Kuta Bali, Indonesia, 8–10 August 2017; pp. 1–4.
5. Mohammadi, M.; Chu, B.-T.; Lipford, H.R. Automated detecting and repair of cross-site scripting vulnerabilities. *arXiv* **2018**, arXiv:1804.01862.
6. Yan, X.-X.; Wang, Q.-X.; Ma, H.-T. Path sensitive static analysis of taint-style vulnerabilities in PHP code. In Proceedings of the 2017 IEEE 17th International Conference on Communication Technology (ICCT), Chengdu, China, 27–30 October 2017; pp. 1382–1386.
7. Parameshwaran, I.; Budianto, E.; Shinde, S.; Dang, H.; Sadhu, A.; Saxena, P. DexterJS: Robust testing platform for DOM-based XSS vulnerabilities. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Bergamo, Italy, 30 August–4 September 2015; pp. 946–949.
8. Wang, R.; Xu, G.; Zeng, X.; Li, X.; Feng, Z. TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting. *J. Parallel Distrib. Comput.* **2018**, *118*, 100–106. [CrossRef]
9. Khalaf, O.I.; Sokiyna, M.; Alotaibi, Y.; Alsufyani, A.; Alghamdi, S. Web attack detection using the input validation method: Dpda theory. *Comput. Mater. Contin.* **2021**, *68*, 3167–3184.
10. Zuhair, H.; Selamat, A.; Salleh, M. Selection of Robust Feature Subsets for Phish Webpage Prediction Using Maximum Relevance and Minimum Redundancy Criterion. *J. Theor. Appl. Inf. Technol.* **2015**, *81*, 188–205.
11. 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]
12. Hosseini, N.; Fakhar, F.; Kiani, B.; Eslami, S. Enhancing the security of patients' portals and websites by detecting malicious web crawlers using machine learning techniques. *Int. J. Med. Inform.* **2019**, *132*, 103976. [CrossRef] [PubMed]

13. Hu, L.; Chang, J.; Chen, Z.; Hou, B. Web application vulnerability detection method based on machine learning. *J. Phys. Conf. Ser.* **2021**, *1827*, 012061. [CrossRef]
14. Malviya, V.K.; Rai, S.; Gupta, A. Development of web browser prototype with embedded classification capability for mitigating Cross-Site Scripting attacks. *Appl. Soft Comput.* **2021**, *102*, 106873. [CrossRef]
15. Mokbal, F.M.M.; Dan, W.; Xiaoxi, W.; Wenbin, Z.; Lihua, F. XGBXSS: An extreme gradient boosting detection framework for cross-site scripting attacks based on hybrid feature selection approach and parameters optimization. *J. Inf. Secur. Appl.* **2021**, *58*, 102813. [CrossRef]
16. Soltani, M.; Siavoshani, M.J.; Jahangir, A.H. A content-based deep intrusion detection system. *Int. J. Inf. Secur.* **2022**, *21*, 547–562. [CrossRef]
17. Pawar, S.; San, O.; Vedula, P.; Rasheed, A.; Kvamsdal, T. Multi-fidelity information fusion with concatenated neural networks. *Sci. Rep.* **2022**, *12*, 5900. [CrossRef]
18. Yang, C.-H.; Pokuri, B.S.S.; Lee, X.Y.; Balakrishnan, S.; Hegde, C.; Sarkar, S.; Ganapathysubramanian, B. Multi-fidelity machine learning models for structure–property mapping of organic electronics. *Comput. Mater. Sci.* **2022**, *213*, 111599. [CrossRef]
19. Guo, M.; Manzoni, A.; Amendt, M.; Conti, P.; Hesthaven, J.S. Multi-fidelity regression using artificial neural networks: Efficient approximation of parameter-dependent output quantities. *Comput. Methods Appl. Mech. Eng.* **2022**, *389*, 114378. [CrossRef]
20. Lu, J.; Lv, F.; Zhuo, Z.; Zhang, X.; Liu, X.; Hu, T.; Deng, W. Integrating traffics with network device logs for anomaly detection. *Secur. Commun. Netw.* **2019**, *2019*, 5695021. [CrossRef]
21. W3Techs. Usage Statistics of Content Management Systems. Available online: https://w3techs.com/technologies/overview/content_management (accessed on 25 September 2022).
22. National Institute of Standards and Technology. National Vulnerability Database. Available online: https://nvd.nist.gov/ (accessed on 25 September 2022).
23. Wireshark. Available online: https://www.wireshark.org/ (accessed on 25 September 2022).
24. duoergun0729. XSS Payloads. Available online: https://github.com/duoergun0729/1book/tree/master/data (accessed on 25 September 2022).