A Source Code Cross-site Scripting Vulnerability Detection Method

Mu Chen[1,2*], Lu Chen[1,2], Zhipeng Shao[1,2], Zaojian Dai[1,2], Nige Li[1,2], Xingjie Huang[3], Qian Dang[4], and Xinjian Zhao[5]
[1] State Grid Smart Grid Research Institute Co.,Ltd,
[2] State Grid Laboratory of Power cyber-Security Protection and Monitoring Technology,
Nanjing 210003, China
[e-mail: 3306940579@qq.com, 46152570@qq.com, shaozhipeng@geiri.sgcc.com.cn,
dzj2002dzj@163.com, 48548962@qq.com]
[3] State Grid Information & Telecommunication Branch,
Beijing 100000, China
[e-mail: xingjie-huang@sgcc.com.cn]
[4] State Grid Gansu Electric Power Company
Beijing 100761, China
[e-mail: 54295925@qq.com]
[5] State Grid Jiangsu Electric Power Company
Nanjing 210003, China
[e-mail: zhaoxj1@js.sgcc.com.cn]
[*]Corresponding author: Mu Chen

Received November 3, 2022; revised January 16, 2023; accepted April 26, 2023; published June 30, 2023

Abstract

To deal with the potential XSS vulnerabilities in the source code of the power communication network, an XSS vulnerability detection method combining the static analysis method with the dynamic testing method is proposed. The static analysis method aims to analyze the structure and content of the source code. We construct a set of feature expressions to match malignant content and set a "variable conversion" method to analyze the data flow of the code that implements interactive functions. The static analysis method explores the vulnerabilities existing in the source code structure and code content. Dynamic testing aims to simulate network attacks to reflect whether there are vulnerabilities in web pages. We construct many attack vectors and implemented the test in the Selenium tool. Due to the combination of the two analysis methods, XSS vulnerability discovery research could be conducted from two aspects: “white-box testing” and “black-box testing”. Tests show that this method can effectively detect XSS vulnerabilities in the source code of the power communication network.

Keywords: Vulnerability detection, XSS(cross-site scripting) vulnerability, Static analysis, Dynamic testing, Webpage attack simulation

This work is supported by Research on Key Technologies of Power Mobile Security Protection oriented to Micro Application Architecture under Grant number 5700-202258200A-1-1-ZN.

1. Introduction

With the continuous development and the increasing expansion of power networks, The security of the network's communication is becoming increasingly significant. Cross-site Scripting Vulnerability(XSS) is one of the network security issues with widespread occurrence [1]and would cause heavy property losses. In the 2021-Top ten vulnerabilities[2] published by OWASP(Open Web Application Security Project) organization, injection attack ranks the top three, and XSS ranks in the top ten. It follows that XSS is still one of the intractable obstacles to the security of power communication networks [3].

The essence of the XSS is to inject malicious scripts into the network, so it belongs to the injection attack. There are two main ways to realize injection operation. First is the "black-box attack." Attacks are conducted by attackers without power network manipulation authority. For example, if there is no filtering mechanism for checking input content, the input tag on the web pages would be used as an XSS injection point.

The second is the "white-box attack". Attackers have the right to manipulate the network content. The malicious script would exist or hide in the webpage code —such as the attributes and events within the tags on the webpages.

The research on XSS vulnerability detection mainly includes static analysis methods, dynamic testing methods, and machine learning methods [3]. "White-box attacks" can be solved through static analysis [4]. In addition, it can combine data flow analysis to achieve a more effective analysis of the source code structure of the web page[5]. The dynamic testing method could test the defense quality of web pages by imitating "black-box attacks" [6]to explore potential weaknesses. Machine learning adopts many classification algorithms, such as reinforcement learning and integrated learning algorithms. The detection is completed by building a classification model [7].

Compared with the static analysis method, although the machine learning method could reduce the code audit cost, there remain similarities among the processing procedures. Both rely on inductive features to work, which is why they cannot respond to vulnerability changes at the right moment [3]. The dynamic analysis method can detect the malignant content of the web page in the actual running environment by constructing attack vectors and combining network flow analysis [8]and the packet analysis method. Also, it could monitor the web page status in real time through the web page detection tool. Hence, the dynamic way performs better in handling constantly changing attack ways.

From analyzing black-box and white-box attacks, an XSS detection method, which combines dynamic testing methods and static analysis methods, is proposed. Section 2 of this paper introduces related work, the existing scheme, and the detection method mixed with dynamic and static analysis. Section 3 gives the overall structure of the vulnerability detection model and describes the implementation of each module. Section 4 provides the detection effect and the comparative experimental results. Finally, it is summarized in Section 5.

2. Related Work

There exist many research schemes, coming from the aspects of static analysis, dynamic testing, and machine learning methods, to solve XSS vulnerabilities in the communication networks for the power industry.

Rathore et al. [9]extracted URL, HTML, and SNS from the social network to construct features and used ten-fold cross-validation to process and extract features. Ten classifiers achieve XSS vulnerability detection. It has good accuracy. Wang et al. [10]used the static analysis method to scan browser listing permissions, script content, background pages, and CSS files and summarized many key features. Then they input the features into the “sklearn” classification framework to build a vulnerability detection model. However, machine learning detection methods are highly dependent on data features and cannot deal with constantly changing vulnerabilities. Tariq et al. [11]used the 30 features proposed by Zhou and Wang [12]and applied the basic genetic algorithm to detect malicious XSS payloads. After the training of the feature dataset, the accuracy of “feature confirmation” is as low as 5.78%, and the accuracy of “feature script” is as low as 69.60%. After that, the model is trained by updating the attack vector in the training data and using reinforcement learning. Accuracy is improved. But the model complexity is high.

Gu et al. [13]completed dynamic analysis by decomposing malignant loads into load units, then combining load units in coding to generate targeted loads for dynamic detection. According to the HTML static markup mechanism, Hou et al. [14]marked webpage tags and scripts respectively from the perspective of “black-box testing”. Then, they construct the attack vectors by combining the tag content and finally realize vulnerability detection during the web attack simulation. Their method yields good accuracy.

Through the technical research and analysis of the existing scheme. We concluded that the critical step of vulnerability detection is that detection should be conducted during running the program. Meanwhile, the static analysis could be carried out before running the program to better understand the source code's structure, parts, and security degree. Therefore, this paper proposes a detection method combining dynamic testing and static analysis method. Set the attack vector and malignant content feature expression as the detection tool. Inspired by the genetic algorithm [15][16], we construct the attack vectors with payload elements [17][18]and deformation features which are similar to the mutation operation in the genetic algorithm. The feature expressions are constructed based on the malignant content feature analysis. As a matching tool for malignant content, feature expression could reduce the cost of building a malignant feature database.

3. XSS Vulnerability Detection Model

This chapter shows the implementation details of the vulnerability detection model, including the statement of the overall architecture of the model and the roles of each module. Also, the implementation of the static analysis method and the dynamic test method in the model is described, and the detection standards of the two ways are given.

3.1 Model Structure

The vulnerability detection model constructed in this paper could detect three types of XSS vulnerabilities: "Reflective-XSS," "Stored-XSS," and "DOM-XSS." The detection model comprises preprocessing module, content extraction module, detection tool construction module, and vulnerability detection module. The execution flow of each module is shown in **Fig. 1**. Detecting Tools, storage structures, and some operations in the detection model are displayed as symbols. The details are shown in **Table 1**.

Table 1. Symbol definition table

Symbol	**Description**
F_E	Feature Expression
Payload	The attack vectors
ICN	Tag Token: id-class-name
VariM	The matrix store the information on variables
TagM	The matrix store the information of HTML tags
InputM	The matrix store the information of HTML input elements
ES	The Execute_script() function in the Selenium
DocumentOP	Actions within the page using the Document keyword

1） **Preprocessing module**: There may be confusion operations in the source code of web pages to avoid security review. The existence of confusion makes source code analysis impossible or incorrect. Therefore, preprocessing operations are set up to format the source code to eliminate most of the confusion. Preprocessing operations include:
1. use lowercase letters except for built-in global objects and built-in functions
2. decode Unicode encoding content and URL encoding content
3. clear closed complete comments within the source code
4. replace consecutive spaces with single spaces and eliminate the tab character at the beginning of the line

2） **Content extraction module**: The content extraction module is responsible for extracting the critical code of the web page. The following three parts will be extracted: web input elements such as "input," and "textarea" tags, HTML tags vulnerable to injection attacks, and web script code.

3） **Detection tool building module**: Attack vectors and feature expressions are critical tools for model vulnerability detection. The details of its construction will be described in sections 3.2 and 3.3.

4） **Vulnerability detection module:** The details of the vulnerability detection module will be described in Section 3.4.

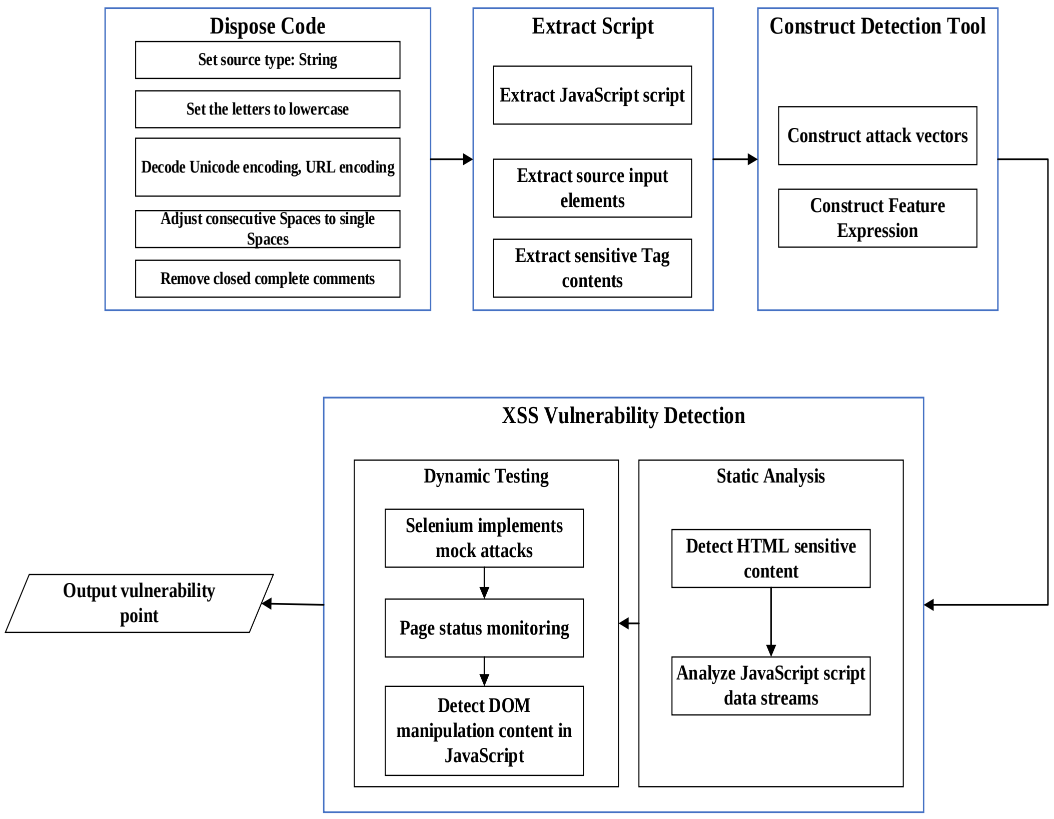

Dispose Code
Set source type: String
Set the letters to lowercase
Decode Unicode encoding, URL encoding
Adjust consecutive Spaces to single Spaces
Remove closed complete comments
Extract Script
Extract JavaScript script
Extract source input elements
Extract sensitive Tag contents
Construct Detection Tool
Construct attack vectors
Construct Feature Expression
XSS Vulnerability Detection
Dynamic Testing
Selenium implements mock attacks
Page status monitoring
Detect DOM manipulation content in JavaScript
Static Analysis
Detect HTML sensitive content
Analyze JavaScript script data streams
Output vulnerability point

Fig. 1. Model structure flow chart

3.2 Static Analysis

The purpose of static analysis technology is to complete the analysis of web page source code. The source code of the analyzed web page includes HTML content and script content such as JavaScript. This section will describe the static analysis process, give the static analysis of vulnerability detection criteria and feature expression construction process, and describe the source code script analysis process.

3.2.1 Static Analysis of Vulnerability Detection Standards

Static analysis completes the identification of malignant content in the HTML source code by utilizing the feature expression "F_E." The content matched by "F_E" is the vulnerability in the HTML code, and the details of the vulnerability, such as the "ICN," will be output.

3.2.2 Feature Expressions Construction

The syntax of "F_E" inherits the syntax of Regular Expressions. "F_E" comprises metacharacters, frequency characters, and content characters. Among them, metacharacters and frequency characters inherit the syntax of Regular Expressions. The content character, with {tag, attribute, event} as its outer content, {pseudo-protocol, script} as its inner content, and {closed character, special symbol} as its supplementary content is in common with the elements which are utilized to construct attack vectors.

3.2.3 Script Analysis

For the dynamic content on the web page, this paper designs a "Variable Conversion" method for data flow analysis. All variables, built-in functions, output functions, and assignment operations in the script were tagged by the "Variable Conversion" method. The objects tested during the script analysis are variable values, function arguments, and the result of the "DocumentOP" operation. For objects with malicious content, their location, value, and data flow will be output. If the variable passes its value to the HTML tag, the details of the HTML tag would also be output.

3.3 Dynamic Testing

The purpose of dynamic testing technology is to simulate the website under attack to measure the presence of vulnerabilities in the webpage. This section describes the dynamic testing process. The vulnerability detection standard of dynamic testing and the construction process of attack vectors are given.

3.3.1 Dynamic Detection Vulnerability Detection Standard

The dynamic testing method is implemented by Selenium. With the Selenium browser drive, we could write programs simulating attacks and real-time monitor the status of a web page. For example, monitoring the presence of an "Alert" object to determine whether popover is triggered on the webpage. Further inspection of the properties of the Alert object determines whether the attack vector injected into the page was executed successfully. Thus, we can find out whether the web page has reflective XSS. The presence of DOM-type XSS on the page is determined by taking the contents of the web page path and the console output and using "F_E" for malignancy detection. The steps of "Stored-XSS" detection includes the detection steps of "reflected-XSS" vulnerability and also verify the database content.

3.3.2 Attack Vectors Construction

The attack vector is the critical content of the simulation attack and the core tool of the dynamic test method. According to the set succession relationship, six basic elements: *{tag, attribute, pseudo protocol, script, event, and closed character}* will be utilized to construct the attack vector. The inheritance relationships between base elements are shown in **Fig. 2**. Seven deformation methods: *{case confusion, character encoding, blank character replacement, closed symbol replacement, popover closure coincidence change, mark nesting, and adding special symbol}* will be added to the attack vector to enhance further its ability to evade the defense mechanism. The deformation features are combined according to the set collocation mode and then added to the attack vector.

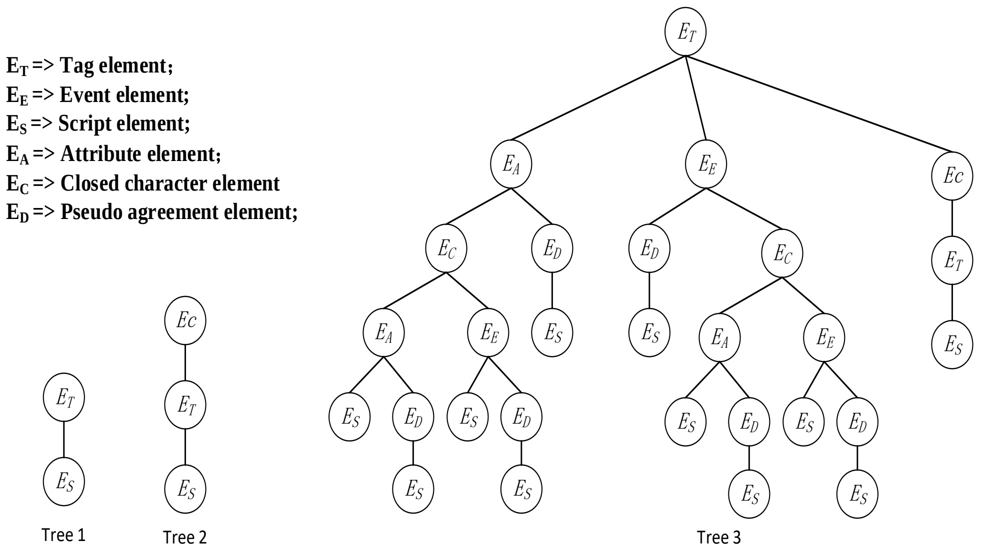

E_T => Tag element;
E_E => Event element;
E_S => Script element;
E_A => Attribute element;
E_C => Closed character element
E_D => Pseudo agreement element;
Tree 1
Tree 2
Tree 3

Fig. 2. Attack vector basic element succession Figure

This method constructs attack vectors by Selenium dynamic manipulation of HTML files. The specific flow chart is shown in **Fig. 3**.

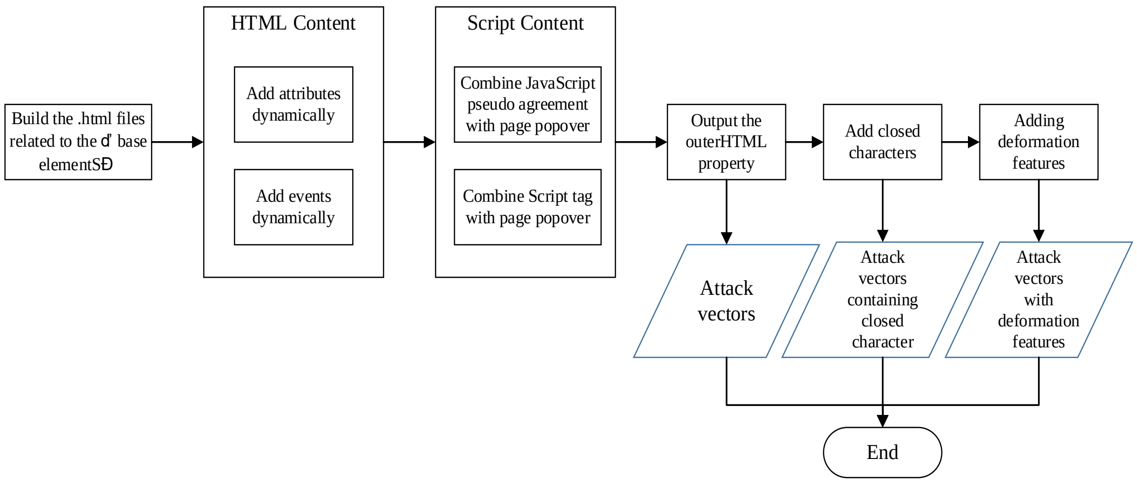

Build the .html files related to the ď base elementSĐ
HTML Content
Add attributes dynamically
Add events dynamically
Script Content
Combine JavaScript pseudo agreement with page popover
Combine Script tag with page popover
Output the outerHTML property
Add closed characters
Adding deformation features
Attack vectors
Attack vectors containing closed character
Attack vectors with deformation features
End

Fig. 3. Attack vector construction flow chart

3.4 Vulnerability Detection Process

Vulnerability detection is based on the source code of web pages, including HTML static content and dynamic content of scripts. The specific detection steps are shown in **Table 2**.

Table 2. Vulnerability Detection Procedures

Vulnerability detection model execution steps	
Data pre-processing	① unified lowercase format ② decode the encoded content ③ remove comments ④ remove excess Spaces and tabs
Data content extraction	① input element extraction ② sensitive label extraction ③ script extraction and variable content extraction
Vulnerability detection	
Static Analysis	While **index < TagM.length** do ①AttrDe: detect attribute content.②EventDe: detect event content； IF **AttrDe** || **EventDe == True** ①Output the content of this row in the matrix. ②Output tag's ICN While **index** < **VariM.length** do ①**ParaDe**: detect the parameter of the output function If **ParaDe == True** Output the detailed content of the function ①**VariDe**: detect the value of the variable If **VariDe == True** ①Output variable, variable value, variable location ②Output the target that contains the contents of this variable, including the tag's outerHTML attribute、variable location
Dynamic Testing	① load web pages or files with Selenium. ② Inject attack vectors to input elements. ③ **PageStatusContent**: monitor the page's popover status, and match the popover's content with the content of the attack vector. ④**EsContent**: get the DocumentOP value with ES function IF **PageStatusContent** || **EsContent == True** do Output the content of the input element

4. Experiment

This chapter mainly states the experimental test results of the model. Firstly, the environment configuration of the experiment is given, then the test results of each function of the model and the comparative experimental results are presented.

4.1 Experimental Environment Configuration

The specific environment configuration for the test is shown in **Table 3**.

Table 3. Test configuration table

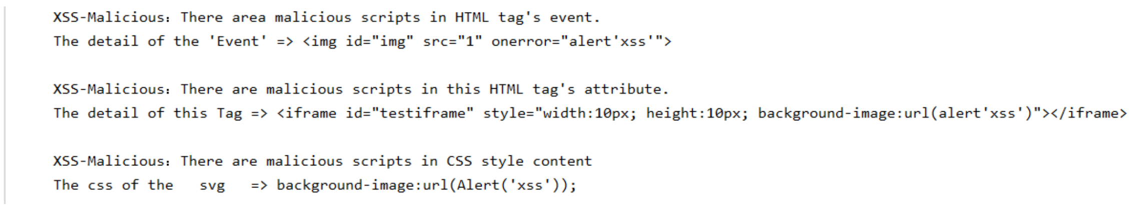
Operating System	Windows10
Central Processing Unit	AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz
Development Language	Python3.8
Development Kits	JetBrains Pycharm 2019
Database	MySQL

4.2 Experimental Test Results

This experimental section mainly presents the ability of the detection method to detect XSS vulnerabilities. The detection tool for this article is named SMD. One of the experimental subjects is a web page set containing elements for external input, and the other is two vulnerability platforms -- DVWA and Pikachu. Among them, the former could set up vulnerability defense mechanisms of different security levels, so it can effectively measure the ability of attack vectors and indirectly reflect the vulnerability and security of web pages. The experimental content of this paper will be described from two aspects: static analysis and dynamic testing.

4.2.1 Static Analysis

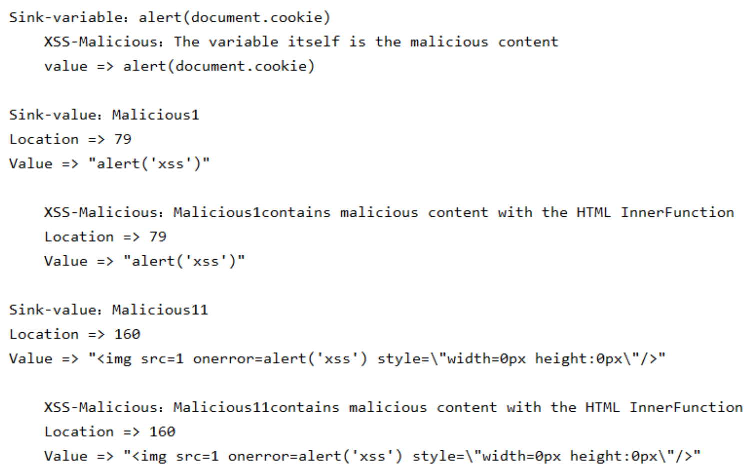
The static analysis includes HTML code analysis and script code analysis. In the HTML code analysis, the suspicious attributes and events would be captured, and malicious content detection for the threatening content would be conducted. For labels with malicious content, detailed information about the tag is displayed. The HTML code analysis results are shown in **Fig. 4**.



```
XSS-Malicious: There area malicious scripts in HTML tag's event.
The detail of the 'Event' => <img id="img" src="1" onerror="alert'xss'">

XSS-Malicious: There are malicious scripts in this HTML tag's attribute.
The detail of this Tag => <iframe id="testiframe" style="width:10px; height:10px; background-image:url(alert'xss')"></iframe>

XSS-Malicious: There are malicious scripts in CSS style content
The css of the   svg   => background-image:url(Alert('xss'));
```

Fig. 4. Static analysis of HTML content detection results

To analyze script contents, all variables, “DocumentOP,” built-in functions, output functions, assignment operations, and value transfer operations were captured by the “Variable Conversion” method, and the caught content was stored in the matrix “VariM”. Then, we traverse the “VariM” matrix. Identify whether a function or assignment exists threatening content and conduct malignancy detection for threatening content. Through this step, we successfully discovered the threat and potentially malicious content in the script. The detection effect is shown in **Fig. 5**.

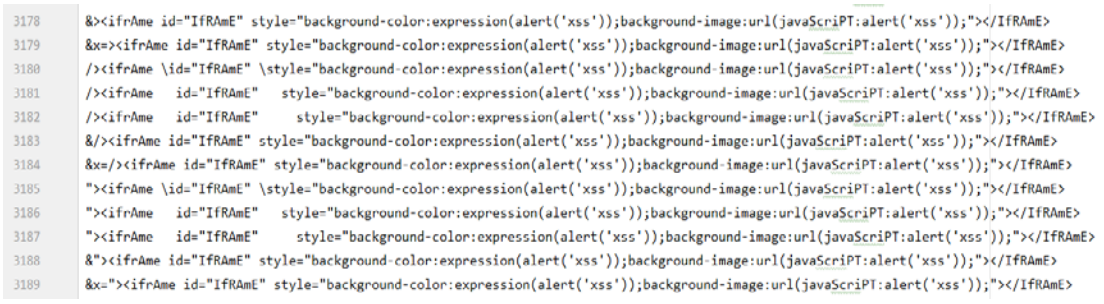


```
Sink-variable: alert(document.cookie)
    XSS-Malicious: The variable itself is the malicious content
    value => alert(document.cookie)

Sink-value: Malicious1
Location => 79
Value => "alert('xss')"

    XSS-Malicious: Malicious1contains malicious content with the HTML InnerFunction
    Location => 79
    Value => "alert('xss')"

Sink-value: Malicious11
Location => 160
Value => "<img src=1 onerror=alert('xss') style=\"width=0px height:0px\"/>"

    XSS-Malicious: Malicious11contains malicious content with the HTML InnerFunction
    Location => 160
    Value => "<img src=1 onerror=alert('xss') style=\"width=0px height:0px\"/>"
```

Fig. 5. Detection result of the static script analysis

4.2.2 Dynamic Testing

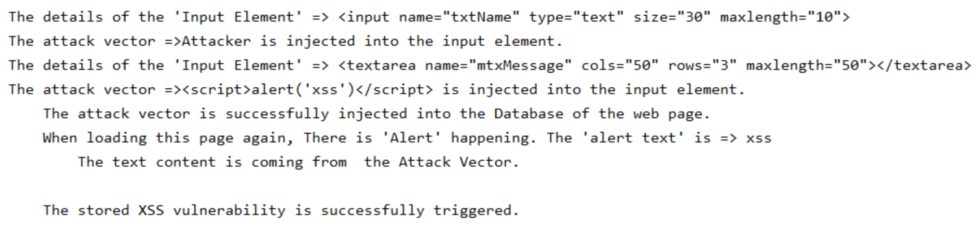
The attack vector is constructed by building HTML files with specific sensitive tags and attributes and dynamically changing tag attributes and events according to various base elements and inheritance relations. Finally, deformation content is added to the attack vector. Seven sensitive tags, eight sensitive attributes, and three accessible triggering events are utilized. The constructed attack vector is shown in **Fig. 6**.



```
&><ifrAme id="IfRAmE" style="background-color:expression(alert('xss'));background-image:url(javaScriPT:alert('xss'));"></IfRAmE>
&x=><ifrAme id="IfRAmE" style="background-color:expression(alert('xss'));background-image:url(javaScriPT:alert('xss'));"></IfRAmE>
/><ifrAme \id="IfRAmE" \style="background-color:expression(alert('xss'));background-image:url(javaScriPT:alert('xss'));"></IfRAmE>
/><ifrAme   id="IfRAmE"   style="background-color:expression(alert('xss'));background-image:url(javaScriPT:alert('xss'));"></IfRAmE>
/><ifrAme   id="IfRAmE"     style="background-color:expression(alert('xss'));background-image:url(javaScriPT:alert('xss'));"></IfRAmE>
&/><ifrAme id="IfRAmE" style="background-color:expression(alert('xss'));background-image:url(javaScriPT:alert('xss'));"></IfRAmE>
&x=/><ifrAme id="IfRAmE" style="background-color:expression(alert('xss'));background-image:url(javaScriPT:alert('xss'));"></IfRAmE>
"><ifrAme \id="IfRAmE" \style="background-color:expression(alert('xss'));background-image:url(javaScriPT:alert('xss'));"></IfRAmE>
"><ifrAme   id="IfRAmE"   style="background-color:expression(alert('xss'));background-image:url(javaScriPT:alert('xss'));"></IfRAmE>
"><ifrAme   id="IfRAmE"     style="background-color:expression(alert('xss'));background-image:url(javaScriPT:alert('xss'));"></IfRAmE>
&"><ifrAme id="IfRAmE" style="background-color:expression(alert('xss'));background-image:url(javaScriPT:alert('xss'));"></IfRAmE>
&x="><ifrAme id="IfRAmE" style="background-color:expression(alert('xss'));background-image:url(javaScriPT:alert('xss'));"></IfRAmE>
```

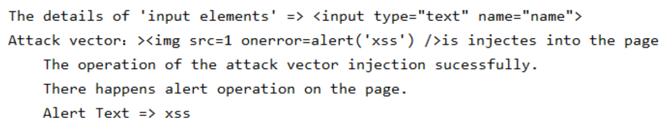
Fig. 6. Attack vectors

After constructing the attack vector, the dynamic attack is conducted on the DVWA and Pikachu platforms. Attack vectors could be injected into the platform with SMD. The web page operation triggered by the attack vector and the content of the operation could be successfully caught by SMD. The detection effect of "Stored-XSS" is shown in **Fig. 7**. The detection effect of "Reflected-XSS" is shown in **Fig. 8**. The webpage URL, URL paths, and other information could be output in real time by SDM. Also, it performs malignant detection on these contents and outputs malignant URLs. The detection effect of "DOM-XSS" is shown in **Fig. 9**. The webpage URL, URL path, and other information are obtained in real-time and detected immediately through SMD. So the malignant URL which would trigger "DOM-XSS" is detected successfully. The detection effect of "DOM-XSS" is shown in Figure 9.



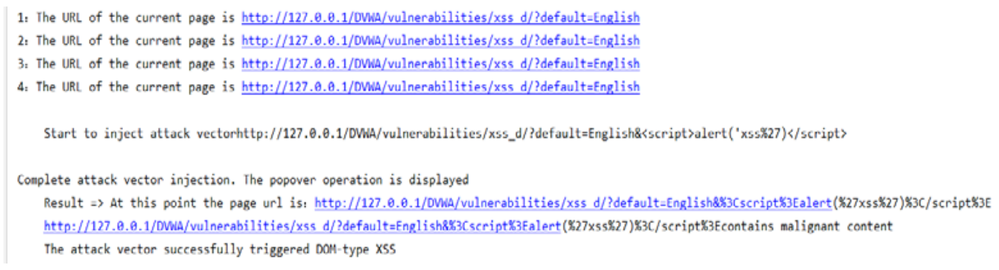
```
The details of the 'Input Element' => <input name="txtName" type="text" size="30" maxlength="10">
The attack vector =>Attacker is injected into the input element.
The details of the 'Input Element' => <textarea name="mtxMessage" cols="50" rows="3" maxlength="50"></textarea>
The attack vector =><script>alert('xss')</script> is injected into the input element.
    The attack vector is successfully injected into the Database of the web page.
    When loading this page again, There is 'Alert' happening. The 'alert text' is => xss
        The text content is coming from  the Attack Vector.

    The stored XSS vulnerability is successfully triggered.
```

Fig. 7. Stored-XSS detection effect



```
The details of 'input elements' => <input type="text" name="name">
Attack vector: ><img src=1 onerror=alert('xss') />is injectes into the page
    The operation of the attack vector injection sucessfully.
    There happens alert operation on the page.
    Alert Text => xss
```

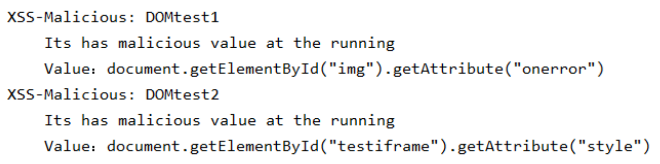
Fig. 8. Reflected-XSS detection effect



```
1: The URL of the current page is http://127.0.0.1/DVWA/vulnerabilities/xss_d/?default=English
2: The URL of the current page is http://127.0.0.1/DVWA/vulnerabilities/xss_d/?default=English
3: The URL of the current page is http://127.0.0.1/DVWA/vulnerabilities/xss_d/?default=English
4: The URL of the current page is http://127.0.0.1/DVWA/vulnerabilities/xss_d/?default=English

    Start to inject attack vectorhttp://127.0.0.1/DVWA/vulnerabilities/xss_d/?default=English&<script>alert('xss%27)</script>

Complete attack vector injection. The popover operation is displayed
    Result => At this point the page url is: http://127.0.0.1/DVWA/vulnerabilities/xss_d/?default=English&%3Cscript%3Ealert(%27xss%27)%3C/script%3E
    http://127.0.0.1/DVWA/vulnerabilities/xss_d/?default=English&%3Cscript%3Ealert(%27xss%27)%3C/script%3Econtains malignant content
    The attack vector successfully triggered DOM-type XSS
```

Fig. 9. DOM-XSS detection effect

Finally, SMD executes "DocumentOP," the rest of the variable matrix, through the "ES" function to detect the returned value. The detection results are shown in **Fig. 10**. Through this step, the correlation between the dynamic test method and the static analysis method is further improved, and the coverage of the static analysis method is also improved.



```
XSS-Malicious: DOMtest1
    Its has malicious value at the running
    Value: document.getElementById("img").getAttribute("onerror")
XSS-Malicious: DOMtest2
    Its has malicious value at the running
    Value: document.getElementById("testiframe").getAttribute("style")
```

Fig. 10. Dynamic script content detection results

4.3 Comparative Experiment

The comparative work in this section includes the analysis of the advantages of the proposed detection method over existing dynamic analysis models, and on this basis, the actual detection capabilities of the models are compared.

4.3.1 Method Advantage Analysis

At present, for the detection of XSS vulnerabilities, the dynamic analysis technology carried out from the actual “black box” perspective is popular. Han[19] et al and Zhao[20] et al used crawler technology and dynamic analysis technology respectively to conduct the dynamic test on the existence of webpage XSS vulnerability(“Han” and “Zhao” represent these models). Dynamic testing technology is an extremely effective means to detect XSS vulnerabilities in Web pages. But for Reflected-XSS, malicious links exist on Web pages. Stored-XSS is triggered by the page after the database contents are fetched under source code execution. DOM-XSS is triggered directly within a web page after malicious content is injected at the injection point. These characteristics are not the core goals of dynamic testing technology, but they are the core work needed to complete the white-box analysis of source code. Therefore, it is necessary to use static analysis technology before dynamic testing technology in this paper, and it has the following advantages.

(1) effectively learn the source structure, such as the source data flow
(2) discover the malignant code in the source code
(3) reduce the preparation before the dynamic test
(4) make the detection work more complete
(5) improve the accuracy of web pages vulnerability detection

In addition, in terms of the implementation of dynamic testing technology, the similarities and differences between SMD and Han and Zhao are shown in Table 4.

Table 4. The similarities and differences of realizing dynamic testing technology

Items	Zhao	Han	SMD
Tools	Selenium	Selenium	Selenium
Excavate injection point	Yes	Yes	Yes
Injection point test method	probe vector	probe vector	attack vector
Set the attack vector syntax	Yes	Yes	Yes
Attack vector construction	manual combination	manual combination	Selenium
Vulnerability detection mode	inject attack vectors	inject attack vectors	inject attack vectors
Vulnerability identification method	monitor webpage state	monitor webpage state	monitor the webpage state and analyze the return value of the “ES” function
Vulnerability source code location	No	No	Yes
Data flow analysis	Yes	No	Yes

Table 4 shows the commonalities and differences in the implementation of dynamic testing techniques between the proposed model SMD and Han and Zhao. Among them, SMD has the advantages of source code vulnerability location and data flow analysis, which are based on static analysis of source code. In addition, the Selenium tool was used to build the attack vector according to the attack vector syntax, which effectively accelerated the construction speed of the attack vector.

4.3.2 Comparative Experimental Results

Firstly, we conduct the vulnerability detection of Zhao and SMD on Firing Range[21] project. The result is shown in **Fig. 11**.

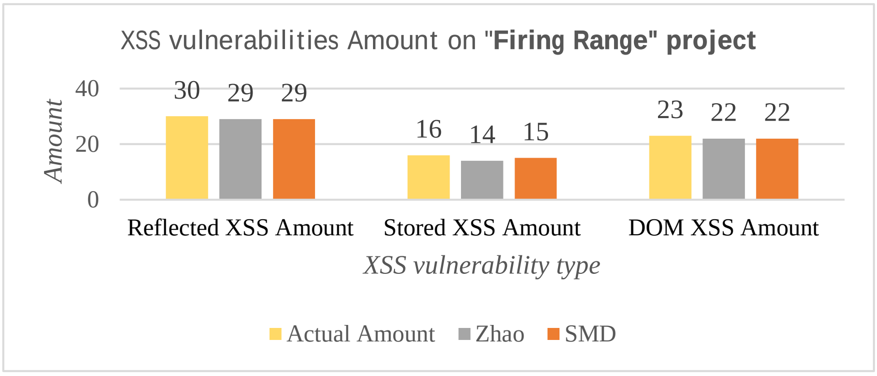


Fig. 11. XSS vulnerability detection comparison

Combined with the amount of XSS vulnerabilities shown in **Fig. 11**, the vulnerability detection Precision and False Alarm Rate of the two models, Zhao and SMD, are shown in **Table 5**. As can be seen from **Table 5**, the combination of static analysis technology in this paper has indeed improved the accuracy of vulnerability detection.

Table 5. Precision and False Alarm Rate of the detection

Model	Number of missed reports	Number of false alarm	Precision	False Alarm Rate
Zhao	6	2	91.30%	2.90%
SMD	4	1	94.20%	1.44%

Secondly, to further test the vulnerability detection capability of SMD, this article compares it with *Acuentix* and *ZAP*, two standard vulnerability detection tools. Both are widely used vulnerability detection tools. We test the web pages, from the *XSSed*[22] website, with XSS vulnerabilities that are not yet fixed. The results are as follows:

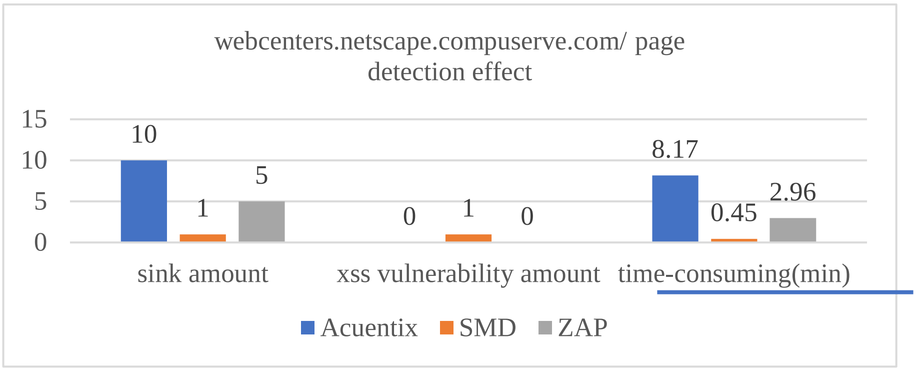


Fig. 12. Comparative experimental results

Fig. 12 shows the results of SMD and two tools detecting a web page. Among them, Less threatening spots were identified by SMD. This is because the tools scan for threat content by fully traversing all links under the site in depth. The SMD model needs to improve the traversal ability of URL links. On the other hand, the vulnerability in the detected web page is that the alert operation outputs “cookie” information. Because *Acuentix* and *ZAP* are used to detect and scan vulnerabilities from a completely black-box perspective, they cannot find malignant content directly. SMD needs to increase the work of deep traversal of URL links in web pages to further improve the comprehensiveness of vulnerability analysis.

Finally, Wang[23] et al. adopted a novel perspective of vulnerability discovery based on event listening, so we compared the detection effects between models on the DVWS vulnerability platform. The target model is labeled “Wang”.

Table 6. Comparative experiments of different models

Target	**SMD**	**Wang**
attack vector construction	Six types of base elements and seven types of deformation characteristics	Attack vector classification feature set and eight deformation features
vulnerability detection scope	stored, reflected, DOM	reflected, DOM
code structure analysis	yes	no
amount of vulnerabilities	3	3
amount of decanting points	4	8

Table 6 shows the detection effects of the two detection models under some indicators. The two models adopt similar ideas to design the construction method of the attack vector. However, the vulnerability detection scope of the SMD model is broader. In addition, SMD uses static analysis to analyze the web source code. Therefore, SMD further improves the comprehensiveness of vulnerability analysis.

5. Conclusion

To ensure the security of the power communication networks. For the cross-site scripting vulnerability in the power grid and the potential security problems of the source code, this paper proposes a cross-site scripting vulnerability detection method from the perspectives of white box and black box attacks, combining the static analysis method and the dynamic testing method to deal with the two types of attacks. The experiments show that the static analysis method can effectively analyze the source code content and detect the malignant content in the code. The dynamic test method could effectively verify the possible degree of web page trigger attack and excavates the injection point information of cross-site scripting vulnerability. The proposed dynamic and static analysis methods have good coverage and accuracy in identifying vulnerabilities.

References

[1] I. Hydara et al., "Current state of research on cross-site scripting (XSS) – A systematic literature review," *Information and Software Technology*, vol. 58, 170–186, pp.177-179, Feb. 2015. Article (CrossRef Link)

[2] http://www.owasp.org.cn/OWASP-CHINA/owasp-project/2021-owasp-top-10/

[3] W. Sun, K. -Y. Zhang, L. -F. Xue, et al., "Overview of XSS vulnerability research," *Information Security Research*, vol. 2, no. 12, pp. 1068-1079, Feb. 2016.

[4] L. K. Shar, H. B. K. Tan, "Automated removal of cross site scripting vulnerabilities in web applications," *Information and Software Technology*, vol. 54, no. 5, pp. 467-478, May. 2012. Article (CrossRef Link)

[5] Z. -Y. Wan and B. Zhou, "Input verification vulnerability detection method based on Static Information Flow Tracking," *Journal of Zhejiang University (Engineering Science)*, vol. 49, no. 4, pp. 683-691, Apr. 2015.

[6] H. Choi, S. Hong, S. Cho and Y. -G. Kim, "HXD: Hybrid XSS detection by using a headless browser," in *Proc. of 2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*, pp. 1-4, Aug. 2017. Article (CrossRef Link)

[7] C. Li, Y. Wang, C. Miao, et al., "Cross-site scripting guardian: A static XSS detector based on data stream input-output association mining," *Applied Sciences*, vol. 10, no. 14, p. 4740, July. 2020. Article (CrossRef Link)

[8] S. Wen, P. Luo, X. Xu et al., "XSS Attack Defense Detection based on Machine learning modeling," *Communications Technology*, vol. 55, no. 3, pp. 351-358, Mar. 2022. Article (CrossRef Link)

[9] S. Rathore, P. K. Sharma, and J. H. Park, "XSSClassifier: An Efficient XSS Attack Detection Approach Based on Machine Learning Classifier on SNSs," *J Inf Process Syst*, vol. 13, no. 4, pp. 1014-1028, Aug. 2017. Article (CrossRef Link)

[10] Yao Wang, Wandong Cai, Pin Lyu, Wei Shao, "A combined static and dynamic analysis approach to detect malicious browser extensions," *Security and Communication Networks*, vol. 2018, May. 2018. Article (CrossRef Link)

[11] I. Tariq, M. A. Sindhu, R. A. Abbasi, A. S. Khattak, O. Maqbool, and G. F. Siddiqui, "Resolving cross-site scripting attacks through genetic algorithm and reinforcement learning," *Expert Systems with Applications*, vol. 168, Apr. 2021. Article (CrossRef Link)

[12] Y. Zhou and P. Wang, "An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence," *Computers & Security*, vol. 82, pp. 261–269, May. 2019. Article (CrossRef Link)

[13] J. -T. Gu and Y. Xin, "XSS vulnerability detection model based on dynamic analysis," *Computer Engineering*, vol. 44, no. 10, pp. 34-41, Nov. 2018. Article (CrossRef Link)

[14] X. -Y. Hou, X. -L Zhao, M. -J. Wu, et al., "A dynamic detection technique for XSS vulnerabilities," in *Proc. of 2018 4th Annual International Conference on Network and Information Systems for Computers (ICNISC)*, IEEE, pp. 34-43, Sept. 2018. Article (CrossRef Link)

[15] C. Cheng, Y. -H. Zhou, "XSS Vulnerability Mining Based on Fuzzy Testing and Genetic Algorithm," *Computer Science*, vol. 43, no. Z6, pp. 328-331, June. 2016. Article (CrossRef Link)

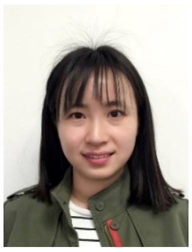
[16] Z. Liu, Y. Fang, C. Huang, et al., "GAXSS: Effective Payload Generation Method to Detect XSS Vulnerabilities Based on Genetic Algorithm," *Security and Communication Networks*, vol. 2022, pp. 3-8, Feb. 2022, Article (CrossRef Link)

[17] Gupta C, Singh R K, Mohapatra A K, "GeneMiner: a classification approach for detection of XSS attacks on web services," *Computational Intelligence and Neuroscience*, vol. 2022, pp. 2-6, May. 2022. Article (CrossRef Link)

[18] Y. Qing, "Research on Attack Vector in XSS Detection," *Southeast University*, no. 5, pp. 23-28, Apr. 2018.

[19] Y. -Y. Han, Y. -R. He, P. -H. Liu, et al., “Design and implementation of XSS vulnerability detection tool based on crawler,” *Journal of Beijing Institute of Electronic Science and Technology*, vol. 27, no. 1, pp.7-16, Mar. 2019. Article (CrossRef Link)
[20] Y. -H. Zhao, D. -Y. Wu, “Analysis and Design of an XSS Vulnerability Detection System,” *Software Guide*, vol. 18, no. 3, pp. 162-167, Jun. 2019.
[21] https://github.com/google/firing-range
[22] http://www.xssed.org/archive/special=1
[23] D. Wang, L. -J. Liu, J. -C. Lin, et al. “XSS Vulnerability detection based on DOM state transition,” *Journal of Beijing University of Technology*, vol. 44, no. 9, pp. 1208-1216, June. 2018. Article (CrossRef Link)

Mu Chen, senior engineer, Director engineer of State Grid Smart Grid Research Institute Co., LTD., head of terminal and access security technology, core backbone of State Grid Corporation Network Security Key Laboratory, with CISP, CISSP, PMP and other professional qualifications, is an authoritative expert in the field of mobile Internet security of State Grid Corporation. As the project leader, independently undertook a number of State Grid Corporation digitization and network security science and technology projects, participated in a number of national projects, and was the leader of a number of State grid Corporation information security key research and development projects.



Lu Chen, female, master, senior engineer, graduated from Nanjing University of Posts and Telecommunications, engaged in network security protection technology

Zhipeng Shao, received his Bachelor's degree in Information and Electronic Engineering from Zhejiang University. Now he is a senior engineer, mainly engaged in the research of power network security system architecture, key technologies and the research and development of special safety protection equipment for power.

Zaojian Dai, male, master, graduated from Harbin Commercial University. He is currently a senior engineer at the Power Grid Digitization Institute of State Grid Smart Grid Research Institute Co., LTD. The research direction is terminal and access information security.

Nige Li, female, master, senior engineer, graduated from Nanjing Normal University, majored in network security protection technology

Xingjie Huang, received his M.S. in Information Assurance from Northeastern University, Boston, Massachusetts, USA, in 2016. In 2017, he started working at State Grid Information & Telecommunication Branch. His area of expertise is cybersecurity, focusing on topics such as cyber attack & defense, data security, cloud computing.

Qian Dang, senior engineer, engaged in network security attack and defense, mobile security, industrial security, Internet security and other security technology research of the power industry.

Xinjian Zhao, Master, graduated from the Department of Computer Science, School of Information Science and Technology, Peking University. Lecturer of Information and Communication Branch of State Grid Jiangsu Electric Power Co., LTD. And First Research Institute of Ministry of Public Security.