Computers & Security 134 (2023) 103434

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

ZTWeb: Cross site scripting detection based on zero trust

Anbin Wu [a], Zhiyong Feng [a], Xiaohong Li [a], Jianmao Xiao [b,c,*]

[a] College of Intelligence and Computing, Tianjin University, Tianjin, China
[b] School of Software, Jiangxi Normal University, Nanchang, China
[c] Jiangxi Provincial Engineering Research Center of Blockchain Data Security and Governance, Nanchang 330022, China

ARTICLE INFO

Keywords:
XSS
Zero trust
Word2Vec
TextCNN
Deep learning

ABSTRACT

Policy defense technology is the mainstream XSS defense technology. However, defense mechanisms with fixed policies may hardly cover the attack surface persistently in dynamic environments. Moreover, the undifferentiated policy makes the malicious code and developer code have the same resource authorization, which leads to the game between the security and usability of the page. To tackle this problem, we propose a zero trust-based defense model - ZTWeb, which constructs differentiated and dynamic policies to balance the security and usability of the website. Specifically, ZTWeb micro-segments the protect surface code into the trust domain, executing different authorization policies based on the trust level of the code subject. The key of ZTWeb is to break the control risk of static policy authorization and create dynamic trust by continuously evaluating the behavior of untrusted domains. Trust evaluation takes the call sequence of sensitive resources as the judgment element. We associate the parent-child relationship between domains and divide the behavior branches within the domain to construct a complete, accurate, context-containing behavior sequence. Furthermore, the extracted sequence is regarded as a piece of text, and the TextCNN model is introduced to identify XSS attacks. We evaluate ZTWeb using real datasets collected from GitHub. The experimental results show that the model can achieve an accuracy of 99.7%, the overall performance overhead is low, and strong security is maintained without destroying the website's usability.

1. Introduction

Although the industry has proposed various defense methods against XSS attacks, high-risk XSS vulnerabilities have been found in the systems of many large enterprises in recent years, such as Microweber (CVE-2022-2495, 2022), TrueConf (CVE-2017-20118, 2017), WordPress (CVE-2022-0234, 2022), IBM (CVE-2021-39068, 2021).

1.1. Introduction to core concepts

To make the article more precise, we first explain the key concepts in this article.

Attack surface. The attack surface is the injection point where an XSS attacker inserts the attack code.

Protect surface. The protect surface comprises the web system's most critical data and applications.

Sensitive data. Sensitive data refers to data related to user privacy in the Web application, such as cookies and tokens.

Sensitive API. Sensitive APIs refer to APIs that can launch XSS attacks, including write-type APIs, request-type APIs, dangerous-type APIs, and sensitive data APIs.

Sensitive resources. Sensitive resources include sensitive data and sensitive APIs.

Trust domain. The code in the trust domain is trustworthy and has authorization for all sensitive resources.

Untrusted domain. The code inside the untrusted domain is untrusted, and the call of sensitive resources requires ZTWeb authorization.

1.2. XSS attack defense and zero trust

Policy defense technology exploits security policies to intercept the injection of XSS attack code and limit its resource authorization. However, the static defense policy is difficult to adapt to the dynamic change of attack means. Moreover, the security policy authorizes the attack code and developer code indiscriminately, resulting in the conflict between higher security and better usability for the target website (Xu et

* Corresponding author at: School of Software, Jiangxi Normal University, Nanchang, China.
E-mail addresses: wuanbin@tju.edu.cn (A. Wu), zyfeng@tju.edu.cn (Z. Feng), xiaohongli@tju.edu.cn (X. Li), jm_xiao@jxnu.edu.cn (J. Xiao).

https://doi.org/10.1016/j.cose.2023.103434
Received 23 June 2023; Received in revised form 26 July 2023; Accepted 13 August 2023
Available online 19 August 2023

al., 2022). Due to the interleaving of the attack surface (Sengupta et al., 2020) and the protect surface (Campbell, 2020), the defense model cannot deploy differentiated policies.

The attack surface is dynamic, complex, and unknown to defenders (Campbell, 2020). On the contrary, the protect surface comprises the most critical data, assets, applications, and services, which are very small, more important, and fully known (Palo Alto Networks, 2018). Thus, the isolation of the protect surface prevents the leakage of core permissions and enables the defense model to adopt differentiated and dynamic authorization policies for different access subjects.

Zero trust (Rose et al., 2020) is based on the principle of "never trust, always verify," (Samaniego and Deters, 2018) breaking the control risk of static policy authorization. As an essential part of zero trust, microsegmentation ensures resource authorization of the protect surface and prevents lateral penetration of the attack surface. Meanwhile, zero trust takes continuous trust evaluation and dynamic authorization (Syed et al., 2022) as essential means, dynamically adjusts authorization policies through continuous monitoring and evaluation of non-protected surface behaviors, and creates a gray "sometimes" area to the traditional black-and-white block-allow access model (Campbell, 2020). Zero trust has become an effective defense mechanism against the XSS attack with dynamic trust and differential policies.

1.3. Our contribution

In this paper, we propose an XSS attack defense model ZTWeb, based on zero trust, which constructs differentiated and dynamic policies to balance the security and usability of the website. The main contributions of this study are as follows:

- Differentiation policy. We isolate the protect surface into the trust domain and divide the attack code into the untrusted domain to adopt different policies for different trust levels.
- Dynamic authorization. We construct a complete and accurate behavior sequence based on the untrusted domain with context information. The continuous trust evaluation takes the behavior sequence as the element of XSS attack detection and dynamically adjusts the authorization policy according to the detection result.
- Extraction of key features. We use the TextCNN model to effectively extract the key features of the behavior sequence and improve the accuracy of XSS attack detection.

The rest of this paper is organized as follows. Section 2 introduces the research work of XSS defense in recent years. Section 3 gives an overview of ZTWeb. Section 4 shows the implementation details of ZTWeb. Section 5 describes the experiment and evaluation of ZTWeb. Section 6 is the summary and future work of this paper.

2. Related work

2.1. Policy defense technology

Policy defense technology uses policy instructions to intercept the injection of attack code and limit its execution ability. Typical defense technologies include CSP (Stamm et al., 2010), BEEP (Jim et al., 2007), Noncespaces (Van Gundy and Chen, 2012), JSCSP (Xu et al., 2022), JSAgents (Heiderich et al., 2015).

CSP (Stamm et al., 2010) is known as a security standard against XSS attacks. W3C recommends that browsers should support CSP policies (W3C, 2018). In CSP1.0, policies restrict the loading of page resources and prohibit the execution of dynamic code through dangerous functions such as eval and settimeout. CSP2.0 allows the execution of the inline script and provides more accurate configuration rules for inline script through nonces and hashes technology.

BEEP (Jim et al., 2007) uses a whitelist to save the hash value of each piece of JavaScript code. The browser compares the hash value of the script segment, and only the code that matches the whitelist can run. BEEP also uses blacklist policies to mark potentially malicious code as no execute, preventing the execution of no execute tagged code at page runtime.

Noncespaces (Van Gundy and Chen, 2012) used randomization techniques to change HTML tags and attributes. The attacker cannot guess the rules of the code, and the constructed code cannot be recognized by the browser. Similarly, Noncespaces also uses policies to configure allowed and prohibited tags and attributes.

JSCSP (Xu et al., 2022) improves CSP's compatibility, scalability, accuracy, and flexibility and provides automatic and accurate policy management. JSCSP uses a whitelist to configure the location of developer code, and the code outside the configuration will be deleted. The read and write of sensitive data and the call of dangerous functions are also intercepted by JSCSP.

JSAgents (Heiderich et al., 2015) mark the tag that violates the policy as "del" before the page is loaded, and the del tag will be deleted during the page loading process. JSAgents configure the key DOM tags write_access and read_access, which limits the reading and modification of data.

Policy defense technology has always been one of the focus directions of XSS attack defense. Security policies are intended to restrict the injection and operation of attack code, but the developer code is also under the control of the policy. Strict policies will lead to the decline of page usability, and loose policies are easily bypassed by XSS attackers, making it challenging to balance usability and security (Xu et al., 2022).

2.2. Machine learning based detection methods

In recent years, with the popularity of machine learning algorithms, more researchers have applied machine learning to improve the efficiency of XSS attack detection. The typical representatives of this method are XSSClassifier (Rathore et al., 2017), DeepXSS (Fang et al., 2018), GraphXSS (Liu et al., 2022), ConvXSS (Kuppa et al., 2022).

XSSClassifier (Rathore et al., 2017) detects XSS attacks on social networking services based on machine learning algorithms. This method extracts 25 features based on URLs, webpage, and SNSs, and applies ten different machine learning algorithms to detect XSS attacks.

DeepXSS (Fang et al., 2018) first preprocessed the input data and then used Word2Vec to extract the feature vector of XSS Payload. Finally, DeepXSS used the LSTM model to identify whether the sequence data was an XSS attack. The experiment showed that the accuracy of DeepXSS could reach 99.5%, and the f1 score was 98.7%.

GraphXSS (Liu et al., 2022) divides the payload data into multiple segments according to the structure of the XSS payload, and the association of segmented data forms a graph structure. The model optimizes the performance of the detection model by combining the graph convolutional network and the residual network. Experiments show that GraphXSS can still achieve good detection results with few samples.

ConvXSS (Kuppa et al., 2022) converts the preprocessed JavaScript code into ASCII coding, and the data of the coding vector after semantic annotation and dimensionality reduction is used as the input of the CNN model for XSS feature extraction and recognition. When the code is detected as an XSS attack, ConvXSS also clears the relevant context code.

Machine learning has achieved a good performance in XSS attack detection. Still, the model training process relies on static code features and ignores dynamic context information (Liu et al., 2022), resulting in false negatives of XSS attacks. Moreover, when the developer code performs the same operation as the malicious code, the machine learning algorithm cannot determine the credibility of the behavior and is prone to false positives.

By analyzing these studies, we find that the mixture of developer code and attack code, as well as the static and indiscriminate nature of the detection policy, causes the false negatives and false positives of

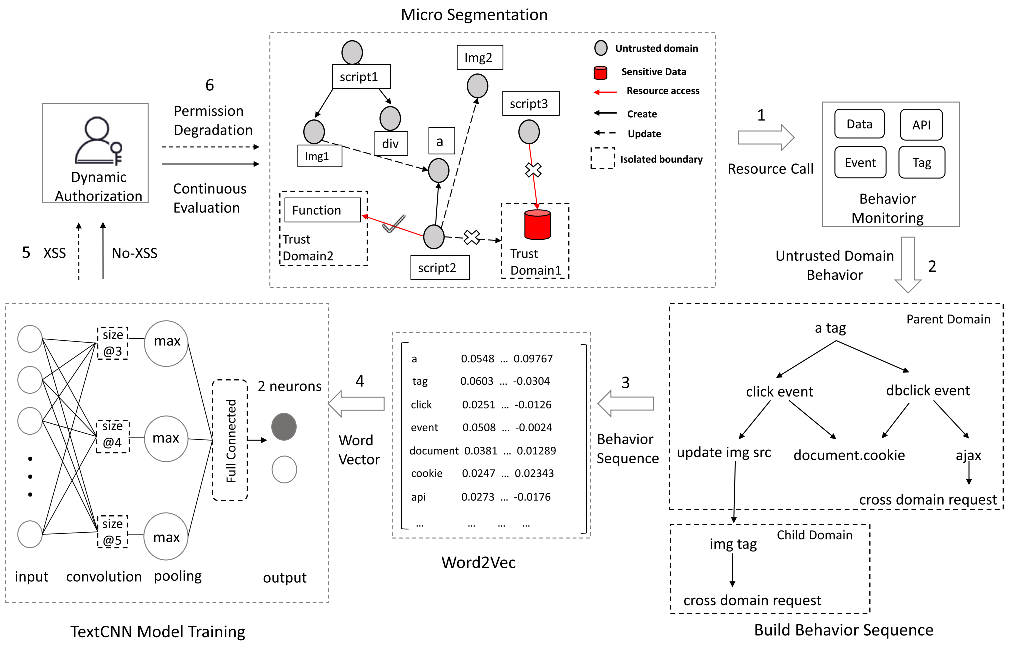

Fig. 1. The Framework of ZTWeb.

XSS attack detection. Therefore, this paper proposes the ZTWeb model based on zero trust, hoping to obtain the balance between usability and security of the website through the isolation of the protect surface, differentiated policies, and continuous trust evaluation.

3. Overview of ZTWeb

The overall framework of ZTWeb is shown in Fig. 1. First, the protect surface is micro-segmented to the trust domain. The injected code and the unprotected surface are separated into the untrusted domain so that they cannot usurp the privileges of the trusted domain. Therefore, ZTWeb can implement differentiated authorization policies based on the trust level of the code subject. Secondly, considering the shortcomings of the static nature of the policy, ZTWeb dynamically adjusts the authorization policy by continuously evaluating the script behavior of untrusted domains. Trust evaluation takes the untrusted domain as the basic unit and the behavior sequence of sensitive data, sensitive API, event, and tag resource call as the judgment element. To improve the accuracy of trust evaluation, ZTWeb uses the parent-child association between domains and the behavior branches within domains to construct a complete, accurate behavior sequence containing context information.

Furthermore, we introduce Word2Vec to convert sequences containing the semantic features of the code into word vectors. The pre-trained word vectors are used as the input of the TextCNN model for feature extraction and classification. Finally, based on classification results, ZTWeb dynamically adjusts resource authorization for untrusted domains.

3.1. Micro-segmentation

The essence of policy-based XSS defense technology is to authorize the user of sensitive resources. However, the mixture of the protect surface and the attack surface results in the uniform authorization of sensitive resources. The protect surface and the attack surface will obtain the permissions of sensitive resources at the same time. In addition, while machine learning techniques can identify XSS attack code, machine learning cannot determine whether the code is trusted when the developer and the attacker implement the same functional code. However, micro-segmentation can take differentiated authorization for different isolation domains and intercept illegal lateral movement between isolation domains. Therefore, the first step in XSS attack defense is micro-segmentation.

ZTWeb isolates the protect surface into the trust domain, enabling authorized code to access sensitive resources while also preventing lateral movement by an attacker (Syed et al., 2022). Codes with different trust levels have different policies, so the usability and security of the page can be balanced.

3.1.1. What is the protect surface of xss attack defense

The protect surface comprises the web system's most critical data and applications. ZTWeb defines the scope of the protect surface according to the following two items: 1). Services based on sensitive data (e.g., cookies, tokens). 2). XSS-like code written by the developer. The following examples show some protect surfaces.

- Reading the cookie makes a cross-domain request.
- Carry the token to initiate the same domain request.
- Same domain or cross-domain requests initiated by the Setinterval API.

3.1.2. Micro-segment the protect surface

ZTWeb isolates the protect surface to the trust domain. First of all, the trust domain should be able to be identified, so the trust level of the domain can be judged during the permission check. Secondly, the trust domain should intercept the penetration of the untrusted domain and prevent the leakage of sensitive resources. Third, the trust domain must still share functions and variables to ensure the page's availability. To meet the above three requirements, the trust domain should follow the following principles:

(1) The principle of unique identification

ZTWeb dynamically assigns random accesstoken identities to trust domains. Domains with invalid tokens are defined as untrusted domains, as shown in Listing 1, the script element and the img tag are isolated to the trust domain.

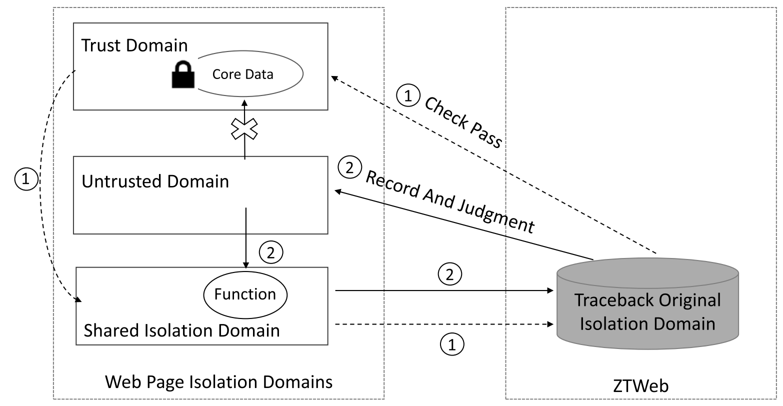

Fig. 2. Traceability of behavior.

Listing 1: Unique identification.

```
<script
    accesstoken="050ed93d3ca311ed9871dce9948ef32c">
Javascript Code
</script>
<img accesstoken="050ed93d3ca311ed9871dce9948ef32c"
    src="a.jpg" onclick="Javascript Code"></img>
```

(2) The principle of protecting sensitive data

The trust domain does not allow sensitive data exposure to other domains as a global variable. Otherwise, the untrust domain can directly obtain the sensitive data without calling the API, making the continuous trust evaluation missing the critical element and affecting the accuracy of the evaluation.

(3) The principle of preventing penetration

An attacker can illegally obtain the permissions of sensitive resources by modifying the trust domain, resulting in a safe escape. Therefore, ZTWeb adds a trust protection mechanism to restrict the untrusted domain from operating on the trust domain, which we will discuss in detail in Section 4.3.

(4) The principle of data sharing

To ensure the sharing of the isolation domain, functions and non-sensitive data can be exposed to other domains. ZTWeb traces the original domain where the function caller is located and executes differentiated policies based on the trust level of the original domain. As shown in Fig. 2, the trust and untrust domains can call the function from other domains. Still, ZTWeb authorizes sensitive resources based on the isolated domain in which the function caller resides.

3.1.3. Separate attack surface

The attack surface is the injection point of the attacker. To prevent the attack surface from stealing the permission of the trust domain, we need to separate the attack surface from the trust domain.

A variable that receives user input is a common injection point for attackers. This attack surface should be separated into the untrusted domain. As shown in Listing 2, the attacker injects code by closing the variable or the script tag. However, the attack code is still in an untrust domain, and its resource authorization still requires continuous trust evaluation.

Injecting attack code through the tag attribute is also an effective way to launch the XSS attack. As shown in Listing 3, the attacker closes the alt attribute and adds a new error event to obtain the permission of the current trust domain. ZTWeb requires the developer to use the setAttribute API to load user input data into the attribute of the trust domain, making it unable to execute.

Listing 2: Script element accept user input.

```
<script>
var variable="<%=userinput%>";
</script>
```

Listing 3: Tag attribute xss attack.

```
<%! String data="xss \"
    onerror=alert(document.cookie) title=\""; %>
<img accesstoken="trust id" src="a.jpg"
    alt="<%=data%>"></img>
```

3.2. Build behavior sequence

Authorizing sensitive resources in untrusted domains can adopt static and dynamic authorization. The traditional static authorization model relies on predefined policies. This static approach fails to adapt to changing conditions while making access decisions (Cheng et al., 2007). Dynamic authorization is characterized by using access policies and estimating dynamic contextual characteristics in real time (Atlam et al., 2017), facilitating effective decision-making based on various data sources. Therefore, we use continuous trust evaluation and dynamic authorization to authorize resources for untrusted domains.

ZTWeb takes the untrusted domain as the basic monitoring unit and continuously records its resource access behavior. The behavior sequence contains the attack intention of the script, which is the core element of continuous trust evaluation.

3.2.1. A complete and accurate sequence of behaviors

By analyzing the XSS attack scripts, We find that the attack behavior is not necessarily initiated by a single domain but may be completed by the parent domain and the child domain. Hence, the behavior analysis based on a single domain is not comprehensive. As shown in Listing 4, the script domain calls the document write API and gets the cookie. The img domain initiates a cross-domain request, and the script domain is its parent domain. If the trust evaluation is initiated in any single domain, the behavior sequence will lack key elements, resulting in evaluation failure.

The inter-domain association ensures the integrity of the behavior sequence. However, the behavior sequence constructed based on all resource calls will produce extensive noise data, thus reducing the accuracy of trust evaluation. Therefore, ZTWeb branches the behaviors in the domain to improve the accuracy of the behavior sequence. ZTWeb divides the script domain into the initialization script branch, tag creation, and tag modification branch and divides the tag domain into the

Listing 4: Isolation domain association.

```
<script>
document.write('<img src=
     "http://hackip/xss?cookie='+document.cookie+'"
     width=0 height=0 border=0/>');
</script>
```

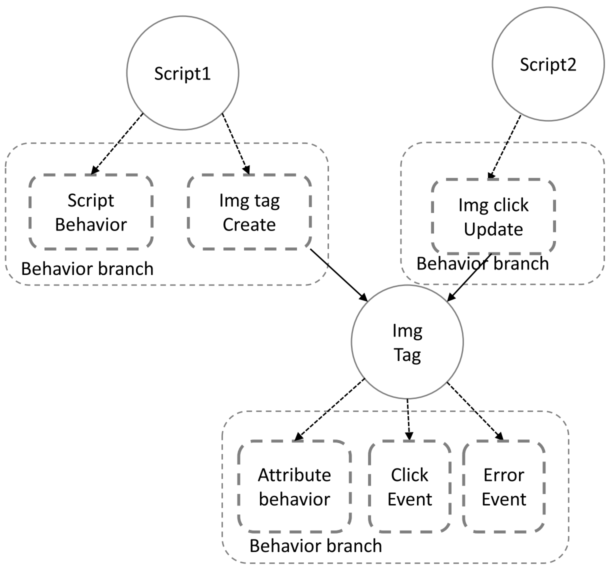

Fig. 3. Untrusted isolation domain graph.

attribute behavior branch and event branch. As shown in Fig. 3, the script1 domain contains a script behavior branch and an img tag creation branch. The img domain has an attribute behavior branch, a click event branch, and an error event branch.

Inter-domain associations mine the dependencies between untrusted domains, and intra-domain branches depict fine-grained behavior trajectories. The untrusted isolation domain graph is formed under the joint action of the two sides. This graph provides the basis for constructing complete, accurate, context-aware behavior sequences.

3.2.2. Context-based resource call sequence

In JavaScript static code detection technology, the syntax unit sequence extracted by abstract syntax tree (AST) cannot extract the semantic relationship of the code. However, the dynamic behavior sequence recognition based on API calls and opcodes has been applied to windows (Ndibanje et al., 2019) and android (Chen et al., 2019) malware detection. A sequence of API calls can be effectively used to model the most representative behavioral features associated with malicious code (D'Angelo et al., 2021).

In the context of the parent domain and behavior branch of the untrusted domain, ZTWeb uses the resource calls of sensitive data, sensitive API, event, and HTML tag to construct the behavior sequence. Sensitive APIs include write-type, request-type, dangerous, and sensitive data APIs. Some APIs are shown in Table 1.

Listing 5 shows the behavior sequence of stealing the cookie: the img tag reads the cookie under an error event and sends the cross-domain request through Ajax API. ZTWeb retains the request type because the same domain request needs to be determined whether it is a CSRF or DDoS attack, and the cross-domain request is more harmful. The type of request is a crucial decision factor. For each resource call, ZTWeb adds a corresponding kind so that the deep learning model can better understand the behavioral semantics, for example, img (tag), error (event).

Table 1
Sensitive API.

Sensitive API	Type
ajax	request
window.open	request
document.write	write
document.createElement	write
eval	danger
settimeout	danger
document.cookie	read sensitive data

Listing 5: Steal cookie behavior sequence.

```
img tag, error event, document cookie api, ajax
     api, send crossdomain request
```

3.2.3. How to break behavior sequence

The behavior in the domain is continuous. If the attack judgment is carried out on the behavior sequence after each behavior occurs, the page's response delay will inevitably increase. Therefore, we need to segment the behavior sequence and launch XSS attack determination when the segmentation point occurs. Since the primary purpose of the XSS attack is to steal sensitive data, CSRF and DDoS, etc., and the end of these attacks is to initiate the same domain or cross-domain requests through the API or tag attribute. So ZTWeb takes the request behavior as a segmentation point, for example, the Ajax API, the src attribute of the img tag.

The behavior sequences of each segment are not wholly unrelated. The new segmented sequences can directly obtain the sensitive data obtained by other segmented sequences or untrusted domains, thus bypassing the trust evaluation of ZTWeb without calling the API. Therefore, ZTWeb adds the acquired sensitive data in the untrusted domain to the new segmented sequence. Although this operation will also lead to misjudgment of the developer code, the untrusted domain has been the leakage of sensitive data. The overall page has been in an unsafe state. The balance should be more focused on the security aspects.

3.3. Feature extraction based on Word2Vec

The behavior sequence contains the behavior pattern of the code. The word embedding model can convert the behavior sequence into a word vector, obtain the semantic features between words, and describe the code's behavior well. Traditional one-hot encoding generates high-dimensional sparse vectors and lacks semantic information between words (Wang and Qian, 2022). Word2Vec (Mikolov et al., 2013) is a language model proposed by Google in 2013, widely used in natural language processing. It can generate low-dimensional dense word vectors while retaining the context information of the vectors, and the basic elements of the sequence do not have polysemy, so we choose Word2Vec to represent the behavior sequence. Word2Vec includes two different training models, the CBOW model, and the Skip-Gram model. The CBOW model can predict the probability of central words according to surrounding words. In contrast, the Skip-Gram model can predict the probability of the surrounding words according to the central word. Considering that Skip-Gram is superior to CBOW in semantic tasks (Mikolov et al., 2013), the model of this paper adopts Skip-Gram.

The framework of Skip-Gram is shown in Fig. 4, which includes three parts: the input layer, the hidden layer, and the output layer. The one-hot encoding of the central word is used as the input layer. Through the projection of the hidden layer weight parameter matrix, $W_{V\times N}$, V represents the vocabulary size in the training sample, and N is the number of neurons. The output layer outputs the context of its specified window size. The hidden layer weight parameter matrix $W_{V\times N}$ obtained by learning and training is the final word vector representation.

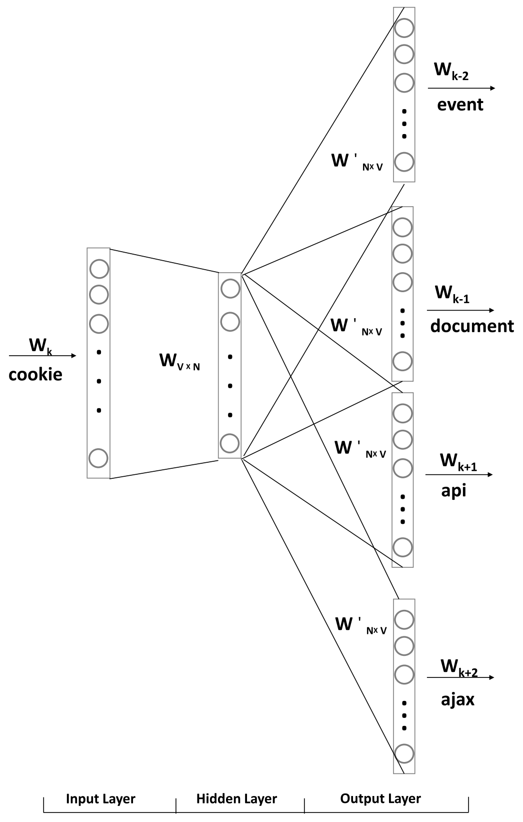

Fig. 4. Skip-Gram model.

3.4. XSS detection based on TextCNN

The extracted behavior sequence can be regarded as a short text, so XSS attack detection becomes a text classification task. Traditional text classification based on machine learning algorithms, e.g., SVM, Decision Tree, or Naive Bayes classifier, requires manual feature extraction and cannot effectively obtain the semantic relationship of behavior sequence, which has certain limitations (Wang and Qian, 2022). Deep learning performs better in text classification tasks, for example, RNN, LSTM, and TextCNN (Kim, 2014). Considering that the constructed behavior sequence is a short text and the low-latency scene of detection, TextCNN has a simple structure, fast training, and retains the semantic relationship between words. Therefore, this paper uses the TextCNN model for behavior sequence classification.

The framework of TextCNN is shown in Fig. 5. We use pre-trained word vectors as input. Branch and segment operations within the domain make the length of the behavior sequence short, so we fix the sequence dimension to 50, truncate the excess part, and fill the zero for the insufficient part. In the convolution layer, we choose a convolution kernel with a size of 3,4,5, which is suitable for capturing the semantic relationship of short texts, followed by the maximum pooling layer, which is connected to a fully connected layer. To prevent overfitting, we use the dropout method. Finally, the probability of each category is output by the softmax function. The specific details of the structure are as follows.

(1) Word embedding

The behavior sequence comprises n resource calls R, and each R is converted into a d-dimensional vector by a pre-trained word vector. Each behavior sequence can be represented by a single-channel N*d matrix. N is the length of the behavior sequence, and zero padding and truncation operations are performed for insufficient and excessive. R_i represents i th resource call in the sequence, and the behavior sequence can be represented by the formula (1).

$$R_{1:n} = R_1 \oplus R_2 \oplus \ldots \oplus R_n \tag{1}$$

(2) Convolution layer

We use three convolution kernels with different window sizes to extract different features of the behavioral sequence. Each window has m filters, including 3 * m filters. The window size is defined as h. The feature t_i extracted in the behavioral sequence $R_{i:i+h-1}$ is shown in formula (2), w is the convolution kernel weight parameter, b is the deviation value, and f is a nonlinear function.

$$t_i = f(w \cdot R_{i:i+h-1} + b) \tag{2}$$

Applying the filter to the matrix, as shown in formula (3), the feature map of the $n-h+1$ dimension is calculated by moving the convolution from top to bottom. 3 * m filters will produce 3 * m different feature maps.

$$t = [t_1, t_2, \ldots, t_{n-h+1}] \tag{3}$$

(3) Pooling layer and fully connected layer

The pooling layer adopts the max pooling. That is, a maximum feature is selected from the feature map of each sliding window, as shown in formula (4).

$$t' = \max(t) \tag{4}$$

The fully connected layer splices each feature of the pooling layer into a feature vector with a length of 3 * m.

(4) Output layer

The feature vector of the fully connected layer is used as the input of the output layer. The SoftMax function predicts the probability that the behavior sequence belongs to each category at the output layer, that is, XSS or No-XSS.

4. Implementation

In this section, we introduce the implementation details of ZTWeb. As shown in Fig. 6, ZTWeb mainly includes four modules, micro-segmentation, protection module, behavior sequence module, and front-end TextCNN model.

4.1. Micro-segmentation

Micro-segmentation is the first step in implementing the ZTWeb model, and we need to follow the isolation principle of the protect surface described in Section 3.

The first is the identification principle. The domain identity attribute accesstoken cannot be guessed by the attacker. Otherwise, it will be used by the attacker to create a trust domain to usurp legitimate authorization. Therefore, accesstoken is randomly created by the server when the page is loaded. Since ZTWeb cannot effectively monitor the behavior of the iframe tag, the applet tag, the object tag, and the embed tag, the above tags must also be isolated to the trust domain.

ZTWeb not only identifies the boundary of the trust domain but also prevents the permission leakage of the trust domain. ZTWeb prohibits defining data as global variables within the trust domain in the following two cases. 1). Sensitive data. The sensitive data of the global variable can make the attacker read the variable directly instead of calling the API, thus escaping the behavior monitoring of ZTWeb. 2). Dangerous variables. A dangerous variable means an attacker can launch an

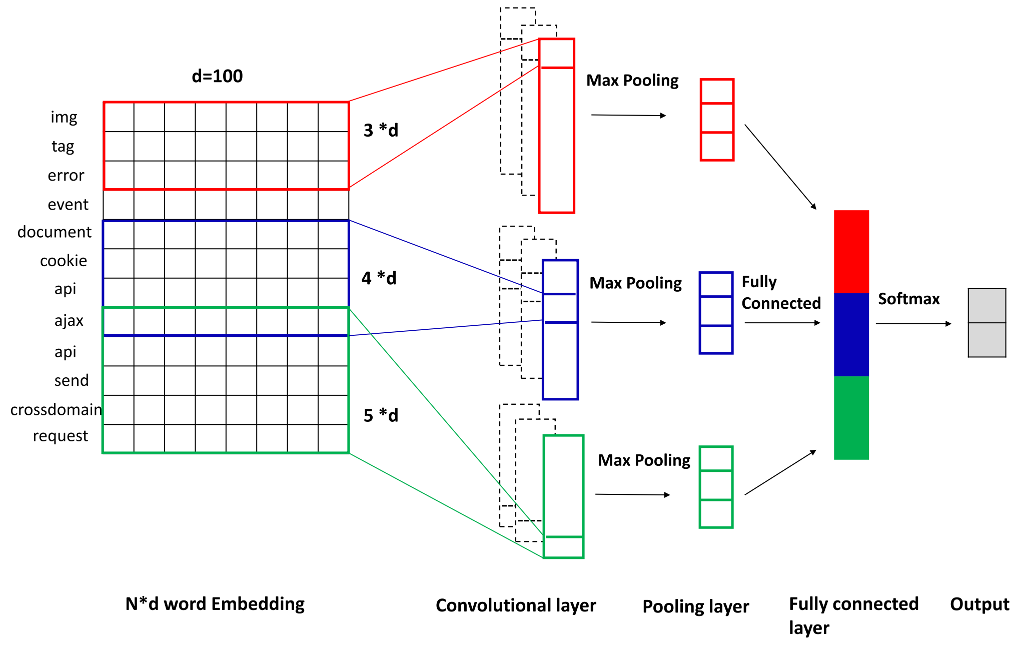

Fig. 5. The framework of TextCNN.

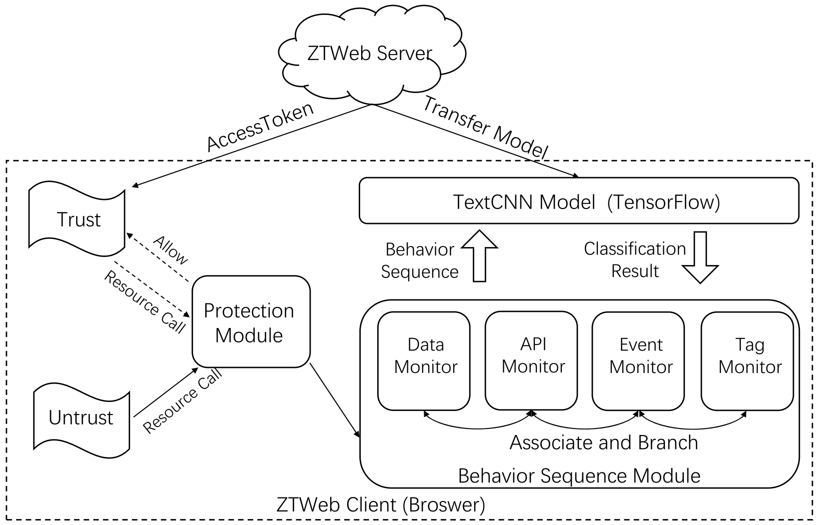

Fig. 6. Implement of the ZTWeb.

XSS attack by modifying the variable. For example, the trust domain sends the cookie data to the specified URL, and the attacker can send the cookie data to his server by modifying the URL. The URL is a dangerous variable. ZTWeb requires developers to use internal variables to define the above two types of variables so that even if an attacker calls a function to launch an attack, ZTWeb will still record its behavior.

4.2. *Behavior sequence module*

ZTWeb implements a dynamic policy for sensitive resource authorization in untrusted domains. We continuously monitor sensitive resource calls in untrusted domains, use the TextCNN model to evaluate the legitimacy of behavior sequences, and dynamically adjust authorization strategies according to the evaluation results. The specific algorithm is shown in Algorithm 1.

(1) Initialize the isolation domain

The isolation domain is the basic unit of ZTWeb. Each domain must be identified to effectively implement inter-domain association and intra-domain branching. We use mutation observer to monitor the creation of HTML tags and set a random and unique uuid for each domain. At the same time, uuid is also set to be unmodifiable to prevent attackers from tampering. Mutation observer also monitors the accesstoken attributes of iframe, applet, object, and embed tags, which have invalid tokens to be cleared to block attacks initiated by data URIs effectively.

Because javascript URI is anonymous, its original domain cannot be traced through window.event(). Therefore, ZTWeb records the anonymous function of javascript URI and determines the execution domain through matching analysis during behavior tracing.

(2) Construct behavior sequence

The behavior sequence of the untrusted domain is generated from two perspectives: 1). completeness, ZTWeb uses the relationship between untrusted domains to ensure the integrity of the behavior sequence. We associate each untrusted domain by hooking the API of

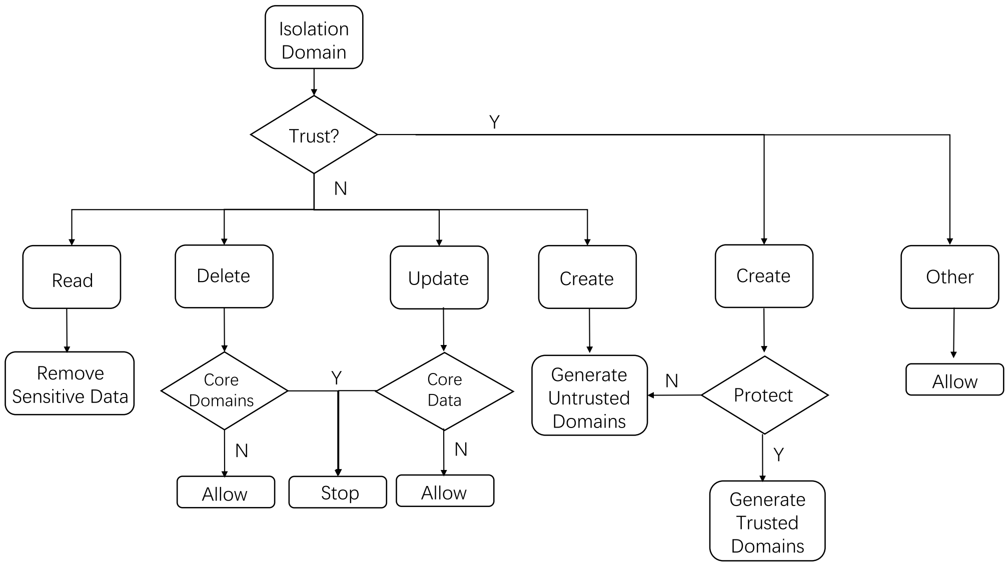

Fig. 7. Protection Module.

the creation type and modification type, and adding and modifying attributes and events. 2). accuracy, ZTWeb first hooks the sensitive function to monitor the call of the API, splits the isolated domain into creation, modification, event, and initialization script branches, and then uses the Object.defineProperty method to record sensitive data requests.

Finally, ZTWeb uses document.currentScript(), window.event(), and anonymous function list to trace the original domain of the resource request. Each untrusted domain will continue to generate a complete and accurate behavior sequence. It should be noted that to prevent the hook API and data from being tampered with, ZTWeb uses the freeze method to configure them as unmodifiable.

(3) Behavior classification

Behavior sequence is the critical element of continuous trust evaluation. ZTWeb uses the TextCNN model to classify behavior sequences. If it is an XSS attack, the authorization of resources will be intercepted. ZTWeb will intercept the authorization of subsequent sensitive resources in the attack domain. When a new behavior occurs, ZTWeb will update the branch's behavior sequence while retaining the previous state of sensitive data acquisition.

Algorithm 1 Construction and recognition of behavior sequence.

```
1: //Initialize Isolation domain
2: setIsolationDomainuuid();
3: deleteInvalidAccessTokenSensitiveTag();
4: recordJavascriptUriFunction();
5: //Monitor behavior
6: apiHook(sensitive_Api_List);
7: dataHook(sensitive_Data_List);
8: domainAssociation();
9: domainBranch();
10: freeze(sensitive_Api_List);
11: freeze(sensitive_Data_List);
12: // Behavior classification
13: getStopOperation()
14: seq = getBehaviorSequene();
15: result = textCNN(seq)
16: if result = = "allow" then
17:     doThisOperation();
18: else
19:     stopThisOpertaion();
20: end if
```

4.3. Protection module

ZTWeb identifies attack behavior and prevents attackers from escaping the monitoring and evaluation of ZTWeb. An attacker can use methods such as parent or innerHTML to obtain the accesstoken of the trust domain and forge the trust domain to launch an attack. An attacker can also usurp the resource permissions of the trust domain by modifying the trust domain.

To reinforce the security of ZTWeb, we propose a trust protection mechanism (Fig. 7), which allows the trust domain to read and modify all domains, and domains dynamically created by the trust domain need to be isolated to the trust domain as long as they meet the definition of the protect surface.

The untrusted domain can read the trust domain but cannot obtain the sensitive data of the trust domain, such as accesstoken, and tokens. ZTWeb removes the sensitive data obtained by calling innerHTML, outerHTML, and other methods in the untrusted domain and prohibits obtaining sensitive attributes through getAttribute API and other methods. Moreover, the permission of the untrusted domain to modify the trust domain is limited, 1). The untrusted domain has no permission to alter the sensitive attributes of the trust domain, ZTWeb hook setAttribute, attribute value, addeventlistener, and other methods to intercept the modification of the critical attributes and events of the trust domain by the untrusted domain. 2). Untrusted domains can delete the trust domain but can not delete the domain of the protection module and behavior sequence module.

4.4. TextCNN model

To reduce the latency caused by server-side model detection, ZTWeb stores the TextCNN model on the browser side. First, ZTWeb uses Python keras to train the TextCNN model on the server side. The trained model can be converted into the format of TensorFlow.js Layers, and ZTWeb loads it to the browser side for attack detection. When the server model is updated, the browser will also initiate the update request to ensure the timeliness of the model.

5. Experiment

This chapter analyzes the model's effectiveness, overload, and comparison with other defense models. The experimental environment is Win10 64-bit operating system, i5-1135G7,8-core processor, 16GB RAM. The server uses Python 3.9 and keras2.9.0 for TextCNN model

Table 2
Confusion matrix.

	Actual XSS	Actual Non-XSS
Predicted XSS	TP	FP
Predicted Non-XSS	FN	TN

training, and the browser uses the TensorFlow.js 3.0 library for browser-side model prediction.

5.1. Dataset

At present, there is no mature browser-side behavior sequence experimental training set. For benign behavior sequences, we collected 50 open-source systems of GitHub, including blogs, wikis, forums, video systems, etc., and extracted the developer behavior sequence of each system. For XSS attack behavior sequences, we use XSS Filter Evasion Cheat Sheet (PortSwigger Research, 2022) and XSS Payload Dataset (Payloadbox, 2020) to construct attack behavior sequences by combining XSS attack patterns (e.g., cookie stealing, DDoS, CSRF). We perform preprocessing operations such as deduplication and standardization on the collected behavioral sequences. The dataset includes 954 benign and 3365 malicious samples, 80% of which are used to train the model and 20% to test.

5.2. Metrics

The ZTWeb model is binary, and the classification results include positive and negative categories. Classification tasks are usually evaluated using some criteria (Fang et al., 2018; Hu et al., 2023; Kuppa et al., 2022; Liu et al., 2022; Wang and Qian, 2022). These criteria include accuracy, precision, recall, and f1 score. However, considering the unbalanced distribution of data sets, this paper uses accuracy and weighted average method to verify the model's effectiveness. The weighted average precision (weightPre), weighted average recall (weightRec), and weighted average F1 (weightF1) are weighted by each category. The weight is the proportion of the number of categories. The specific calculation formula is as follows:

$$Accuracy = \frac{\mathrm{TP} + TN}{TP + TN + FP + FN} \tag{5}$$

$$Precision = \frac{\mathrm{TP}}{TP + FP} \tag{6}$$

$$\mathrm{Recall} = \frac{\mathrm{TP}}{TP + FN} \tag{7}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{8}$$

$$weightP = w_i \sum_{i=1}^{n} P_i \tag{9}$$

$$weightR = w_i \sum_{i=1}^{n} R_i \tag{10}$$

$$weightF1 = \frac{2 * weightP * weightR}{weightP + weightR} \tag{11}$$

The evaluation criteria are calculated based on the confusion matrix in Table 2. TP represents the number of attack samples correctly predicted as XSS attacks, TN represents the number of legitimate samples correctly predicted as legitimate behaviors, FP represents the number of legitimate samples incorrectly predicted as XSS attacks, and FN represents the number of attack samples incorrectly predicted as legitimate behaviors.

5.3. Detection by TextCNN

We trained the Word2Vec model to convert the sequence of behaviors into word vectors. Word2Vec model parameters include size = 100, window = 8 and mincount = 1. The pre-trained word vector is used as the input of the TextCNN model for XSS classification. After parameter adjustment, three convolution kernel has the best effect. Some parameters are configured as follows: kernel_size = 3,4,5, activation = softmax,batch_size = 64, epochs = 50, dropout = 0.2.

Table 3
Result of detection by TextCNN.

Accuracy	Weight Precision	Weight Recall	Weight F1
0.997093	0.997107	0.997093	0.997091

Table 4
Script delay.

Type	ZTWeb		Without ZTWeb
	Trust Domain	Untrusted Domain	
innerHTML	0.123 ms	0.198 ms	0.014 ms
read cookie	0.109 ms	0.186 ms	0.039 ms
document.write	0.181 ms	0.332 ms	0.023 ms
ajax	2.139 ms	11.139 ms	1.313 ms

The experimental results are shown in Table 3. The accuracy of the TextCNN model, weightPre, weightRec, and weightF1 all exceed 0.99. The results show that ZTWeb can effectively detect XSS attacks. The main reasons for the excellent performance of ZTWeb are as follows: 1). ZTWeb isolates the protect surface to the trust domain, and benign samples from the untrusted domain do not read sensitive data or initiate XSS-like attacks. However, XSS attack samples are designed to steal sensitive data and launch XSS attacks. There are apparent differences between the two types of API call sequences, which is why ZTWeb maintains high accuracy. 2) The inter-domain association and intra-domain branch ensure the integrity and accuracy of the behavior sequence, reduce the noise data, and effectively improve the accuracy of XSS attack detection.

5.4. Overhead

To evaluate the performance overhead caused by ZTWeb, we selected Jfinal_CMS 5.1.0 from CVE and changed it to the Jfinal_CMS_ZTWeb version. The test is mainly conducted in three aspects: deployment overhead, JavaScript execution delay, and page loading delay. The page transmission round-trip delay caused by the TextCNN model and accesstoken is not considered here. First, the TextCNN model is loaded from the server when it is first used, and then the model is loaded from the localStorage. Second, the number of trust domains is small, and the round-trip delay caused by accesstoken can be ignored.

Deployment overhead, the number of protect surfaces in the system is tiny. ZTWeb requires the developer to isolate the protect surface to the trust domain, and the trust evaluation of the untrusted domain is done automatically by ZTWeb. Therefore, ZTWeb has a very low deployment overhead for developers.

JavaScript execution delay. ZTWeb continuously records and evaluates the behavior of JavaScript, resulting in script execution delay. We use two web APIs (console.time, console.timeEnd) to calculate the call time of the function. For each test case, we call 1000 times in the trust and untrusted domains, respectively, and calculate the average time. Table 4 shows the call delay of some APIs and sensitive data. In the untrusted domain, ZTWeb clears the sensitive data returned by the innerHTML method and records the behavior of the document cookie and document write API. However, the resource access of the trust domain is authorized, so the delay of the trust domain is lower than that of the untrusted domain. The average delay caused by continuous behavior records is less than 1 ms, which can be ignored. Ajax API is used as the end point of behavior evaluation. ZTWeb uses the TextCNN model to detect XSS attacks on behavior sequences. The delay caused by model

Table 5
Page loading delay.

Type	Average Number	Delay
common tags	858	4.19 ms
iframe tag	1	6.17 ms
img tag	1	10.53 ms

detection is about 10 ms. Assuming the page contains 100 request operations, it will delay about 1 s, which users can accept.

Page loading delay is mainly considered from the following two aspects: 1). In the loading process of iframe, object, embed, and applet, ZTWeb needs to determine the validity of accesstoken. 2) For the request behavior initiated by tag attributes, ZTWeb will conduct an XSS attack judgment. Therefore, the latency of page loading is determined by the total number of tags and the number of tags of iframe, img, etc.

Seven pages of Jfinal_cms_ZTWeb were used in the test. As shown in Table 5, the average number of common page tags was 858. We performed 50 refreshes of the page, and the average loading delay of all common tags was 4.19 ms. The average detection delay of each tag, such as iframe, is 6.17 ms, which does not affect the website's loading speed. For each tag that needs to initiate a behavior determination type (such as img), ZTWeb causes an average evaluation delay of 10.53 ms, within the controllable range.

5.5. Balance between usability and security of the website

In this section, we evaluate the conflicts between the usability and security of pages caused by existing solutions (JSCSP, ConvXSS) from three aspects: sensitive data leakage, CSRF attacks, and DDoS attacks, and how ZTWeb balances the usability and security of the website.

(1) Leakage of sensitive data

JSCSP uses policies to limit the reading and writing of sensitive data (e.g., cookies). However, when the page needs to support the task of cookies, JSCSP's policy allows JavaScript to read cookies, so attackers can use this permission to leak sensitive data.

Listing 6: Vulnerable PHP code.

```
$headerlib->add_js('var zoomToFoundLocation =
     "'.$zoomToFoundLocation.'";');
```

CVE-2016-7394. The page where the vulnerability is located needs to read cookies. The cookie read policy of JSCSP is true. JSCSP also configures the script location whitelist and URL whitelist, but the vulnerability injection point exists in the script whitelist, as shown in Listing 6. After the attack code reads the cookie, it can leak the cookie to the same domain URL. If JSCSP sets the cookie read permission to false, the normal functionality of the page will fail. However, ZTWeb isolates user input data to untrusted domains. Even if an attacker obtains cookies, it cannot send data to the same domain or cross domains.

(2) XSS-based CSRF attacks

The defender uses csrftoken to defend against CSRF attacks. However, if XSS vulnerabilities exist on the page, the attacker can use XSS attacks to steal the current csrftoken and launch CSRF attacks. ConvXSS uses the JavaScript code segment to train the CNN model. Still, the code segment that reads the csrftoken after the click event and initiates the AJAX request in the same domain can be either legal code or attack code, and there will be conflicts during the training of the CNN model.

Listing 7: CSRF attack code or developer code.

```
$(".avatar").on("click", function(){
 var csrfToken = $("[name='csrf']").val();
 $.ajax({
  url:"/post/comment",
  type:"POST",
  data:{"postId":$("#postId").val(),
     "name":$("#name").val(),
     "csrf":csrfToken}
})})
```

Lab:Exploiting XSS to perform CSRF. We implement the demo by imitating PortSwigger's XSS attack on Lab (PortSwigger, 2023). An attacker can inject XSS attack code into the comment page. When other users access the comment page and click the injection button, the attack code will read csrftoken and launch CSRF attacks. As shown in Listing 7, if ConvXSS identifies this type of code segment as an XSS attack during model training, it will lead to false positives of other normal pages. If it is marked as a legitimate code, CSRF attacks will occur in this scenario. ZTWeb isolates the developer code segment that initiates the server request after obtaining the csrftoken to the trusted domain. CSRF attacks created by untrusted domains are intercepted.

(3) XSS-based DDoS attacks

Attackers use setinterval API to launch continuous access to the target system. When the number of victims increases to a certain extent, the continuous access of many clients makes the target system unable to provide regular services, which is the XSS-based DDoS attack. Setinterval API is used by developers to send data requests to the server periodically or to report data periodically. Similarly, attackers can also use setinterval API to launch DDoS attacks. ConvXSS's model training annotation remained ambiguous for this type of code segment.

CVE-2021-46034. In this vulnerability, an attacker can use setinterval to launch DDoS attacks on the same domain or other legitimate domains. As shown in Listing 8, model training at ConvXSS still had problems with security and usability. JSCSP successfully defends this vulnerability with script whitelisting, but the setinterval API is also available on the page. Once injection points are exposed to legitimate script tags or events, JSCSP's configuration policy cannot balance security with usability. ZTWeb isolates the setinterval code block implemented by developers to the trust domain, which guarantees the legitimate authorization of normal functions and intercepts the same domain or cross-domain requests initiated by untrusted domains through setinterval.

Listing 8: DDoS attack code or developer code.

```
setInterval(function() {
  $.get("/path/to/server", function(data, status) {
  });
}, 1000);
```

5.6. Comparison with other solutions

This section compares ZTWeb with other defense techniques against XSS attacks, including CSP, BEEP, JSCSP, GraphXSS, and ConvXSS.

CSP is one of the most popular XSS defense methods. Developers configure policy directives to intercept the execution of XSS attack code. However, the limitations of CSP significantly affect the performance of large Web applications (Weinberger et al., 2011). According to Google's 2017 survey, only 2% of the top 100 Alexa websites support CSP, and 94.72% of CSP policies can be bypassed (The W3C working draft, 2017).

BEEP uses the whitelist of script blocks hash values to block XSS attacks. However, the browser must verify each script block's hash value.

Table 6
Comparison With Other Solutions.

Type	Support DOM-Based XSS	Overhead	Dynamic And Differentiated Policies	Defensive Surface	Balance Between Security And Availability
CSP	Yes	Middle	No	Script Block	No
BEEP	Yes	High	No	Script Block	No
JSCSP	Yes	Low	No	Web Page	No
GraphXSS	No	Middle	No	User Submitted Content	No
ConvXSS	Yes	Low	No	Web Page	No
ZTWeb	**Yes**	**Low**	**Yes**	**Isolation Domain**	**Yes**

The load caused by this operation cannot be ignored, which will increase the loading delay of the front-end page. Meanwhile, BEEP does not support the execution of dynamically generated script blocks on the server side (Niakanlahiji and Jafarian, 2017).

JSCSP controls the resource authorization of the entire page with a more accurate configuration. However, the author of JSCSP also mentioned that the defense scope of the policy is the whole page. If the policy is too strict, the availability of the page will be destroyed. If the policy is too loose, the security of the page will decline.

GraphXSS applies deep learning technology to the field of XSS detection and uses the graph neural network and residual network to improve the accuracy of XSS detection. However, GraphXSS cannot support the detection of DOM-based XSS attacks (Klein, 2005). The detection only depends on the data submitted by the user and does not consider the characteristics of the dynamic context environment. Moreover, GraphXSS seeking to protect all attack surfaces of the system is also considered impractical. (Campbell, 2020).

ConvXSS detects XSS attacks by extracting the JavaScript code and using ASCII encoding to train the CNN model to detect XSS attacks, but the model cannot verify the credibility of the code. When attackers and developers use similar functional code, the model tends to one side, either causing the attack code to take effect or causing the developer code to fail.

Table 6 summarizes the comparison between ZTWeb and other XSS attack defense methods. ZTWeb can support comprehensive XSS attack-type detection under the premise of low overhead. With the support of zero trust technology, it can trace the trust of sensitive resource access behavior and adopt dynamic and differentiated policies for the trust level of isolated domains to grasp the balance between website availability and security.

5.7. Limitations

ZTWeb still has some potential problems. First, window.location() is not allowed to be redefined by JavaScript, so ZTWeb cannot effectively control it, which may lead to a URL redirection attack. Second, ZTWeb does not detect the legitimacy of cross-domain request links. If an attacker does not steal sensitive data and jump directly to a phishing site, ZTWeb cannot capture it. Finally, the TextCNN model is exposed to the attacker, who can invalidate the attack detection by modifying the client model.

6. Conclusions

This paper proposes an XSS detection model based on zero trust – ZTWeb. The model intercepts the attack surface's penetration by isolating the protected surface, which guarantees the developer code's resource authorization. Moreover, ZTWeb continuously monitors the behavior of untrusted domains and uses the TextCNN model to identify whether the behavior sequence is an XSS attack to adjust the resource authorization of untrusted domains dynamically. Experiments show that ZTWeb has a high recognition rate and a low load on the system. Compared with other XSS detection methods, ZTWeb supports more comprehensive XSS detection types and can balance the security and availability of the page.

However, ZTWeb still has shortcomings. In the future, we will modify the browser kernel to monitor the behavior in the domain more comprehensively and accurately. At the same time, the TextCNN model will also be embedded into the kernel to improve the security of the model. For phishing sites, we will use Google Safe browsing (Safe Browsing, 2018) to monitor the safety of jump links in real time.

CRediT authorship contribution statement

Anbin Wu: Conceptualization, Methodology, Software, Writing – original draft. **Zhiyong Feng:** Investigation, Resources, Validation. **Xiaohong Li:** Project administration, Supervision. **Jianmao Xiao:** Data curation, Formal analysis, Writing – review & editing.

Declaration of competing interest

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

Data availability

Data will be made available on request.

Acknowledgements

This work was supported by the National Key R&D Project (2021YFF1201102) and Jiangxi Provincial Natural Science Foundation (No. 20224BAB212015).

References

Atlam, H.F., Alenezi, A., Walters, R.J., Wills, G.B., 2017. An overview of risk estimation techniques in risk-based access control for the Internet of Things. In: 2nd International Conference on Internet of Things, Big Data and Security, pp. 1–7.

Campbell, M., 2020. Beyond zero trust: trust is a vulnerability. Computer 53 (10), 110–113. https://doi.org/10.1109/MC.2020.3011081.

Chen, T., Mao, Q., Lv, M., Cheng, H., Li, Y., 2019. Droidvecdeep: Android malware detection based on word2vec and deep belief network. KSII Trans. Int. Inf. Syst. 13 (4), 2180–2197.

Cheng, P.-C., Rohatgi, P., Keser, C., Karger, P.A., Wagner, G.M., Reninger, A.S., 2007. Fuzzy multi-level security: an experiment on quantified risk-adaptive access control. In: 2007 IEEE Symposium on Security and Privacy (SP'07), pp. 222–230.

CVE-2017-20118, 2017. TrueConf cross-site scripting vulnerability. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-20118. (Accessed 21 July 2023).

CVE-2021-39068, 2021. IBM cross-site scripting vulnerability. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-39068. (Accessed 21 July 2023).

CVE-2022-0234, 2022. WordPress cross-site scripting vulnerability. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-0234. (Accessed 21 July 2023).

CVE-2022-2495, 2022. Microweber cross-site scripting vulnerability. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-2495. (Accessed 21 July 2023).

D'Angelo, G., Ficco, M., Palmieri, F., 2021. Association rule-based malware classification using common subsequences of API calls. Appl. Soft Comput. 105, 107234. https://doi.org/10.1016/j.asoc.2021.107234.

Fang, Y., Li, Y., Liu, L., Huang, C., 2018. DeepXSS: cross site scripting detection based on deep learning. In: Proceedings of the 2018 International Conference on Computing and Artificial Intelligence, pp. 47–51.

Heiderich, M., Niemietz, M., Schwenk, J., 2015. Waiting for CSP – securing legacy web applications with JSAgents. In: Pernul, G., Ryan, Y.A., Weippl, E. (Eds.), Computer Security – ESORICS 2015. Springer International Publishing, pp. 23–42.

Hu, T., Xu, C., Zhang, S., Tao, S., Li, L., 2023. Cross-site scripting detection with two-channel feature fusion embedded in self-attention mechanism. Comput. Secur. 124, 102990. https://doi.org/10.1016/j.cose.2022.102990.

Jim, T., Swamy, N., Hicks, M., 2007. Defeating script injection attacks with browser-enforced embedded policies. In: Proceedings of the 16th International Conference on World Wide Web, pp. 601–610.

Kim, Y., 2014. Convolutional neural networks for sentence classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1746–1751.

Klein, A., 2005. DOM based cross site scripting or XSS of the third kind. Web Appl. Secur. Consortium, Articles 4, 365–372.

Kuppa, K., Dayal, A., Gupta, S., Dua, A., Chaudhary, P., Rathore, S., 2022. ConvXSS: a deep learning-based smart ICT framework against code injection attacks for HTML5 web applications in sustainable smart city infrastructure. Sustain. Cities Soc. 80, 103765. https://doi.org/10.1016/j.scs.2022.103765.

Liu, Z., Fang, Y., Huang, C., Han, J., 2022. GraphXSS: an efficient XSS payload detection approach based on graph convolutional network. Comput. Secur. 114, 102597. https://doi.org/10.1016/j.cose.2021.102597.

Mikolov, T., Chen, K., Corrado, G.S., Dean, J., 2013. Efficient estimation of word representations in vector space. In: International Conference on Learning Representations.

Ndibanje, B., Kim, K.H., Kang, Y.J., Kim, H.H., Kim, T.Y., Lee, H.J., 2019. Cross-method-based analysis and classification of malicious behavior by API calls extraction. Appl. Sci. 9 (2). https://doi.org/10.3390/app9020239.

Niakanlahiji, A., Jafarian, J.H., 2017. WebMTD: defeating web code injection attacks using web element attribute mutation. In: Proceedings of the 2017 Workshop on Moving Target Defense, pp. 17–26.

Palo Alto Networks, 2018. Define a protect surface to massively reduce your attack surface. https://www.paloaltonetworks.com/blog/2018/09/define-protect-surface-massively-reduce-attack-surface/. (Accessed 21 July 2023).

Payloadbox, 2020. Payloadbox/XSS-payload-list: cross site scripting (XSS) vulnerability payload list. https://github.com/ismailtasdelen/xss-payload-list. (Accessed 21 July 2023).

PortSwigger Research, 2022. Cross-site scripting (XSS) cheat sheet. https://portswigger.net/web-security/cross-site-scripting/cheat-sheet. (Accessed 21 July 2023).

PortSwigger, 2023. Lab: exploiting XSS to perform CSRF. https://portswigger.net/web-security/cross-site-scripting/exploiting/lab-perform-csrf. (Accessed 21 July 2023).

Rathore, S., Sharma, P.K., Park, J.H., 2017. XSSClassifier: an efficient XSS attack detection approach based on machine learning classifier on SNSs. J. Inf. Process. Syst. 13 (4), 1014–1028. https://doi.org/10.3745/JIPS.03.0079.

Rose, S., Borchert, O., Mitchell, S., Connelly, S., 2020. Zero Trust Architecture. Special Publication (NIST SP). National Institute of Standards and Technology, Gaithersburg, MD.

Safe Browsing, 2018. Google safe browsing. https://safebrowsing.google.com. (Accessed 21 July 2023).

Samaniego, M., Deters, R., 2018. Zero-trust hierarchical management in IoT. In: 2018 IEEE International Congress on Internet of Things (ICIOT), pp. 88–95.

Sengupta, S., Chowdhary, A., Sabur, A., Alshamrani, A., Huang, D., Kambhampati, S., 2020. A survey of moving target defenses for network security. IEEE Commun. Surv. Tutor. 22 (3), 1909–1941. https://doi.org/10.1109/COMST.2020.2982955.

Stamm, S., Sterne, B., Markham, G., 2010. Reining in the web with content security policy. In: Proceedings of the 19th International Conference on World Wide Web, pp. 921–930.

Syed, N.F., Shah, S.W., Shaghaghi, A., Anwar, A., Baig, Z., Doss, R., 2022. Zero Trust Architecture (ZTA): a comprehensive survey. IEEE Access 10, 57143–57179. https://doi.org/10.1109/ACCESS.2022.3174679.

The W3C working draft, 2017. Content security policy level 3. https://www.w3.org/TR/CSP3/. (Accessed 21 July 2023).

Van Gundy, M., Chen, H., 2012. Noncespaces: using randomization to defeat cross-site scripting attacks. Comput. Secur. 31 (4), 612–628. https://doi.org/10.1016/j.cose.2011.12.004.

W3C, 2018. Content security policy 1.0. https://www.w3.org/TR/CSP1/. (Accessed 21 July 2023).

Wang, Q., Qian, Q., 2022. Malicious code classification based on opcode sequences and textCNN network. J. Inf. Secur. Appl. 67, 103151. https://doi.org/10.1016/j.jisa.2022.103151.

Weinberger, J., Barth, A., Song, D., 2011. Towards client-side HTML security policies. In: Proceedings of the 6th USENIX Conference on Hot Topics in Security, p. 8.

Xu, G., Xie, X., Huang, S., Zhang, J., Pan, L., Lou, W., Liang, K., 2022. JSCSP: a novel policy-based XSS defense mechanism for browsers. IEEE Trans. Dependable Secure Comput. 19 (2), 862–878. https://doi.org/10.1109/TDSC.2020.3009472.

Anbin Wu was born in Shijiazhuang, China, in 1990. He received the master's degree in University of Electronic Science and Technology of China, Chengdu, China, in 2016.

He is currently pursuing the Ph.D. degree in software engineering with Tianjin University, Tianjin, China. His current research interests include web security, moving target defense and zero trust.

Zhiyong Feng was born in 1965. He received the Ph.D. degree from Tianjin University, Tianjin, China, in 1996.

He is currently a Professor with the College of Intelligence and Computing, Tianjin University, Tianjin, China. He has authored more than 200 articles. His research interests include service computing, knowledge engineering, and software engineering.

Dr. Feng is a distinguished member of China Computer Federation (CCF), a member of the Association for Computing Machinery (ACM), and the Chairman of ACM China Tianjin Branch. He has served as General Chair and Program Committee Chair of numerous international conferences, such ICWS, ICSS, APWeb-WAIM, etc.

Xiaohong Li received the Ph.D. degree in computer application technology from Tianjin University, Tianjin, China, in 2005. She is a Full Tenured Professor with the Department of Cyber Security, College of Intelligence and Computing, Tianjin University, Tianjin, China. Her current research interests include knowledge engineering, trusted computing, and security software engineering.

Jianmao Xiao received his Ph.D. from the College of Intelligence and Computing, Tianjin University. He is an assistant professor at Jiangxi Normal University. His main research interests include service computing and intelligent software engineering.