Cross-Site Scripting Attacks and Defensive Techniques: A Comprehensive Survey

Article *in* International Journal of Communications, Network and System Sciences · August 2022

DOI: 10.4236/ijcns.2022.158010

1 author:

Sonkarlay J. Y. Weamie
Hunan University
6 PUBLICATIONS **4** CITATIONS

SEE PROFILE

Int. J. Communications, Network and System Sciences, 2022, 15, 126-148
https://www.scirp.org/journal/ijcns
ISSN Online: 1913-3723
ISSN Print: 1913-3715

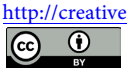
Cross-Site Scripting Attacks and Defensive Techniques: A Comprehensive Survey*

Sonkarlay J. Y. Weamie

Independent Researcher, College of Computer Science & Electronic Engineering, Hunan University, Changsha, China
Email: skweamie@hnu.edu.cn

How to cite this paper: Weamie, S.J.Y. (2022) Cross-Site Scripting Attacks and Defensive Techniques: A Comprehensive Survey. *Int. J. Communications, Network and System Sciences*, **15**, 126-148.
https://doi.org/10.4236/ijcns.2022.158010

Received: June 20, 2022
Accepted: August 6, 2022
Published: August 9, 2022



Abstract

The advancement of technology and the digitization of organizational functions and services have propelled the world into a new era of computing capability and sophistication. The proliferation and usability of such complex technological services raise several security concerns. One of the most critical concerns is cross-site scripting (XSS) attacks. This paper has concentrated on revealing and comprehensively analyzing XSS injection attacks, detection, and prevention concisely and accurately. I have done a thorough study and reviewed several research papers and publications with a specific focus on the researchers' defensive techniques for preventing XSS attacks and subdivided them into five categories: machine learning techniques, server-side techniques, client-side techniques, proxy-based techniques, and combined approaches. The majority of existing cutting-edge XSS defensive approaches carefully analyzed in this paper offer protection against the traditional XSS attacks, such as stored and reflected XSS. There is currently no reliable solution to provide adequate protection against the newly discovered XSS attack known as DOM-based and mutation-based XSS attacks. After reading all of the proposed models and identifying their drawbacks, I recommend a combination of static, dynamic, and code auditing in conjunction with secure coding and continuous user awareness campaigns about XSS emerging attacks.

Keywords

XSS Attacks, Defensive Techniques, Vulnerabilities, Web Application Security

1. Introduction

XSS (Cross-Site Scripting) is a programming-related flaw [1] that occurs when

*Cross-site scripting attacks.

user input data is not correctly sanitized. The attacker exploits this vulnerability to inject unfiltered scripting code into the web application, resulting in account takeover, session or cooking stealing, and rerouting to the attacker's website when the parser processes the script [2] [3]. XSS attack can be initiated on any susceptible website written in any programming language.

A thorough analysis of Cross-Site Scripting vulnerabilities has been presented in detail. We talked about what XSS is, the numerous forms of XSS assaults, how an attacker may exploit this weakness, the results of an XSS attack, and the protective strategies established by the research community to fight against XSS attacks. On the other hand, we examined those defensive strategies and identified the shortcomings in how they were defended against particular XSS attacks.

However, despite researchers' efforts, XSS attacks [4] can still disrupt web applications at a larger skill irrespective of the fact that various tactics and approaches for preventing vulnerabilities have been established. Due to the virtually unchanged browser behavior, it is difficult to detect XSS attacks and differentiate between malicious JavaScript and legitimate online content.

Several sections of the paper are precisely organized according to their respective topics: The definition and classification of XSS, as well as the injection methods utilized by XSS and the damage it causes to web-based applications, are covered in Segment 2. Segment 3 describes the research data composition and compares the CWE Names using the software development vulnerability data for analysis. Segment 4 presents the related work. Segment 5 discusses the XSS prevention and defense mechanism along with the researchers' defensive techniques for XSS attacks (advantages & disadvantages). Segment 6 describes the challenges associated with detecting and defending against XSS attacks along with the precise precautionary measures that should be implemented in response to a given episode. The current issues are broken down into their parts, and then the perspective for the future is presented.

2. Background of the Cross-Site Scripting Attack

2.1. Categories of XSS Attacks

A cross-site scripting attack generally occurs when an attacker compromises a website by inserting malicious JavaScript code into the client-side input parameters. **Figure 1** depicts a comprehensive perspective of the four XSS attack scenarios covered in this paper.

XSS vulnerability exploits [5] the fact that web applications execute scripts in user browsers. If a user tampers with or alters a dynamically generated script, it puts an online application in danger. Although there are four categories of XSS attacks mentioned in this paper, as illustrated in **Figure 1**, most contemporary web application developers and researchers are familiar with only three of them since they are more common in the research community [6] [7]. Organizations such as the Open Web Application Security Project (OWASP) [1] [8] have recognized these three types of XSS attacks as the most common XSS attack vectors on the web.

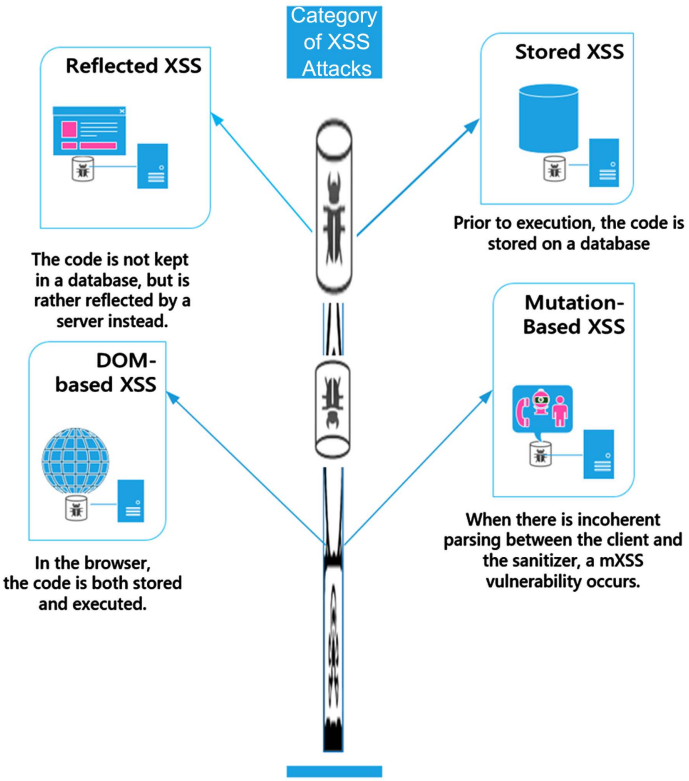


Figure 1. A brief overview of the four categories of cross-site scripting vulnerabilities.

Although each of the four categories of attacks takes a somewhat different approach to exploiting web applications, they are still geared toward the same end goal of collecting user account information as generally illustrated in Figure 2.

However, if you're not familiar with XSS attacks, this should help put things into perspective. As indicated in Figure 1 regarding the four categories of XSS assaults and also displayed in Figure 2 depicting the typical circumstances of XSS attack vector, the following details about the aforementioned categories are explained respectively.

2.2. Stored Cross-Site Scripting (XSS) Attack

This form of XSS vulnerability is sometimes referred to as a persistent XSS. This is due to the fact that the malicious script is still present on the server after the attack has been completed [9]. During this type of attack, the attacker injects code that has been maliciously written onto the server in such a way that it cannot be removed. As shown in Figure 3, the scenario I used to illustrate a stored XSS attack [10] injected a script tag directly into the Document Object Model (DOM) and subsequently executed a malicious script using JavaScript hypothetically. However, while this is the most popular method of exploiting XSS, it is also the most common approach neutralized by advanced security professionals and security-conscious software developers [11]. A user uploads a malicious XSS script to a database, requested and viewed by other users, resulting in script execution on their systems as described in Figure 3.

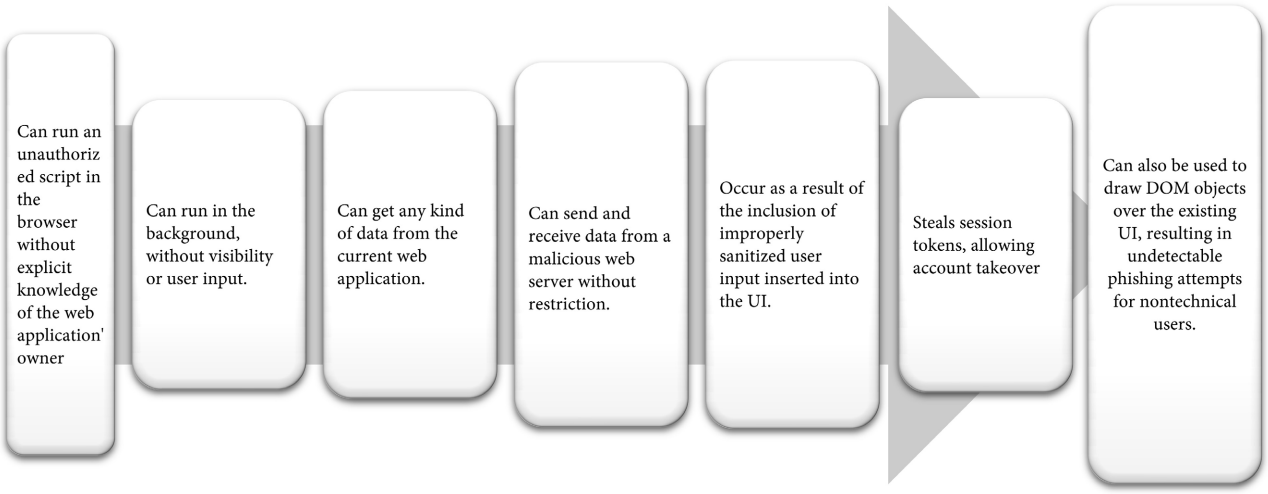


Figure 2. Injection methods of a typical cross-site scripting attacks.

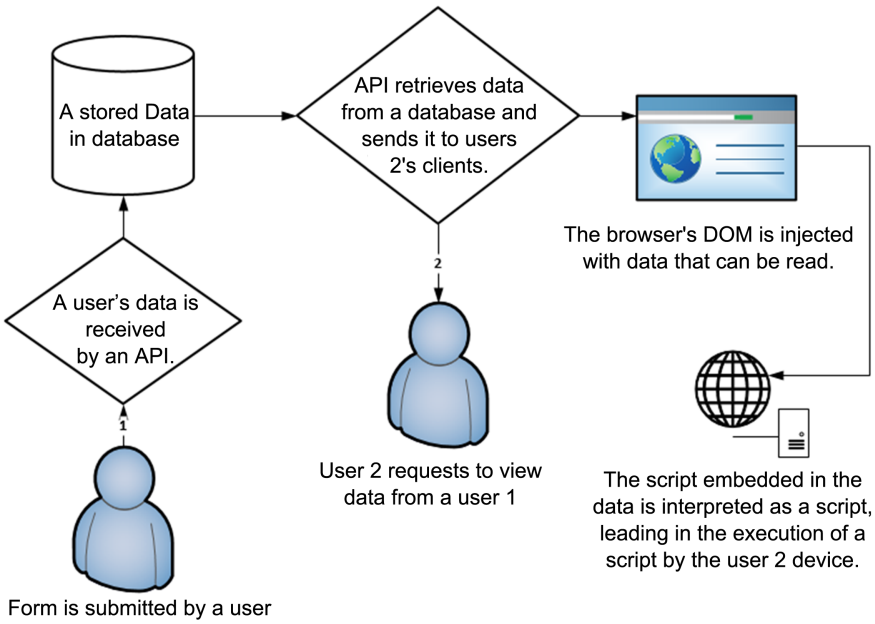


Figure 3. Stored XSS attack scenario.

2.3. Reflected Cross-Site Scripting (XSS) Attack

A reflected XSS attack, which is also known as a non-persistent attack, is where the attacker generates a URL that injects arbitrary scripts into the target web application [12]. Most publications and academic resources introduce reflected XSS before moving on to stored XSS concepts. I feel that reflected XSS attacks are frequently more difficult for newly inexperienced programmers to discover and exploit than stored XSS attacks [13].

A stored XSS attack is relatively straightforward to comprehend from a developer's perspective. The client provides a resource to the server, which is commonly done through the HTTP protocol. The server inserts the requested resource into a database after receiving it from the client. The malicious script

will then be executed unintentionally inside the client's internet browser if other clients later access that resource, as shown in Figure 3.

On the other hand, reflected XSS attacks work like stored XSS attacks but don't require a database or a server. No server is involved in a reflected XSS attack because the client code is affected directly in the browser, as demonstrated in Figure 4. Web applications can be vulnerable to this type of attack (see Figure 4) because of actions taken by a user that executes an unstored (interconnected) script on the user's computer.

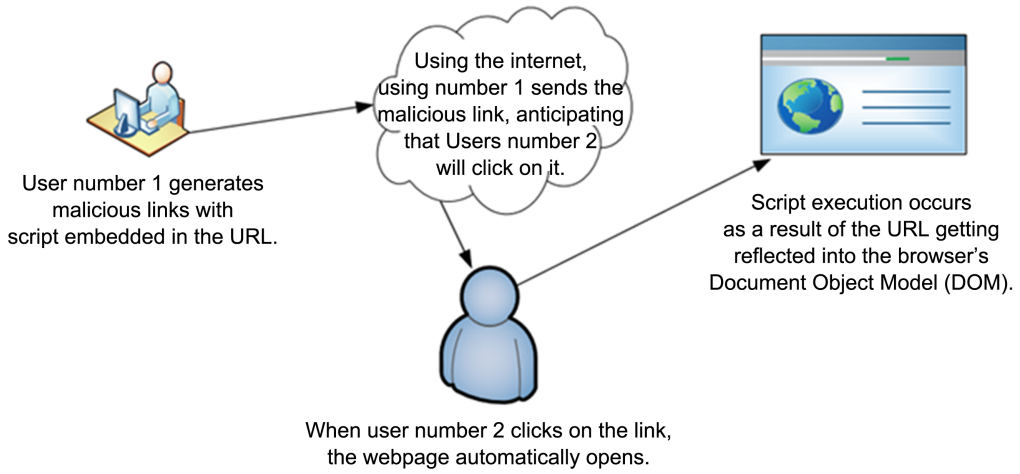


Figure 4. Reflected cross-site scripting scenario.

2.4. Document Object Model-Based Cross-Site Scripting (XSS) Attack

The DOM-based XSS attack [12] [14] is obviously a client-side attack. The DOM-based XSS attack type is depicted in Figure 5 as the third important classification for XSS attacks.

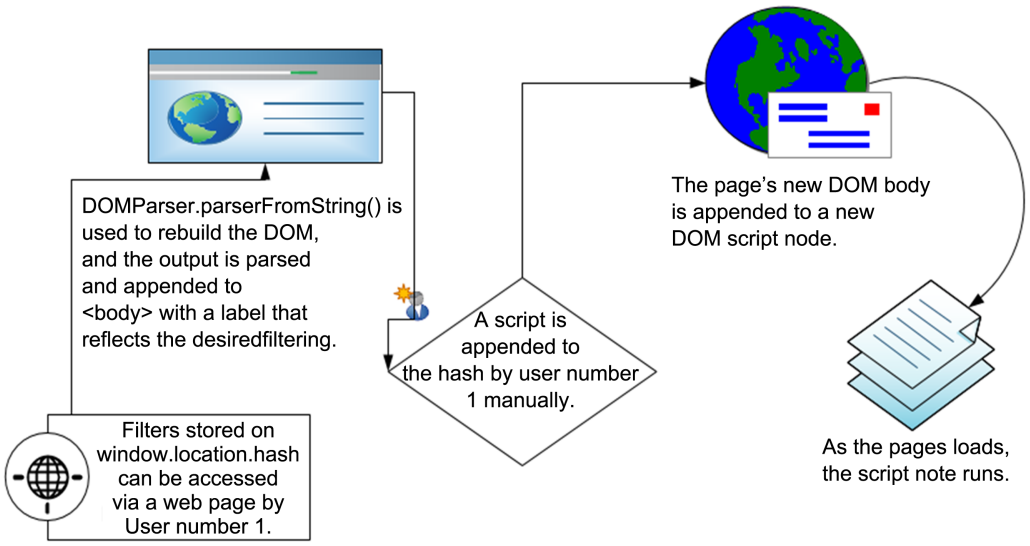


Figure 5. Dom-based cross-scripting attack scenario.

The implementation of the DOM in different browsers may make some browsers vulnerable, while others may not. Compared to typical reflected or stored XSS attacks (Figure 3 & Figure 4), these XSS attacks require an extensive understanding of the browser's DOM and JavaScript to be discovered and exploited.

The DOM-based XSS attacks [15] are principally distinct from other types of XSS in that they do not necessitate communication with a server in any way. As a matter of convention, the source is typically a DOM object that can store text, and the sink generally is a DOM API that can execute a script that has been stored as text.

Both the "source" and the "sink" must be present in the browser's DOM in order for DOM XSS [16] to work because there's no server involvement. In most cases, the sink is a DOM API that can run a script stored in the source as text. It's nearly impossible to detect DOM XSS with static analysis tools or other popular scanners because it never touches a server [17].

2.5. Mutation-Based Cross-Site Scripting (mXSS) Attack

Dr. Mario Heiderich unveiled six (6) new mXSS attack sub-classes in his publication [18]. In mXSS attack, the DOM can be avoided entirely by using InnerHTML, which enables automatic changes to be made to the HTML content. mXSS is sometimes referred to as mutated XSS or mutation-based XSS. This is due to the fact that it is difficult to predict and involves recursion. When the HTML script is loaded into the browser's Document Object Model, the data is mutated, which causes an error. However, the content loaded into the browser's DOM is mutated to verify that it is error-free and does not include any improper markup. This is accomplished by using the element.innerHTML attribute. The fundamental downside of this form of XSS attack is its ability to circumvent server-side defenses and client-side filters. Figure 6 depicts a potential scenario for mutation-based XSS attacks.

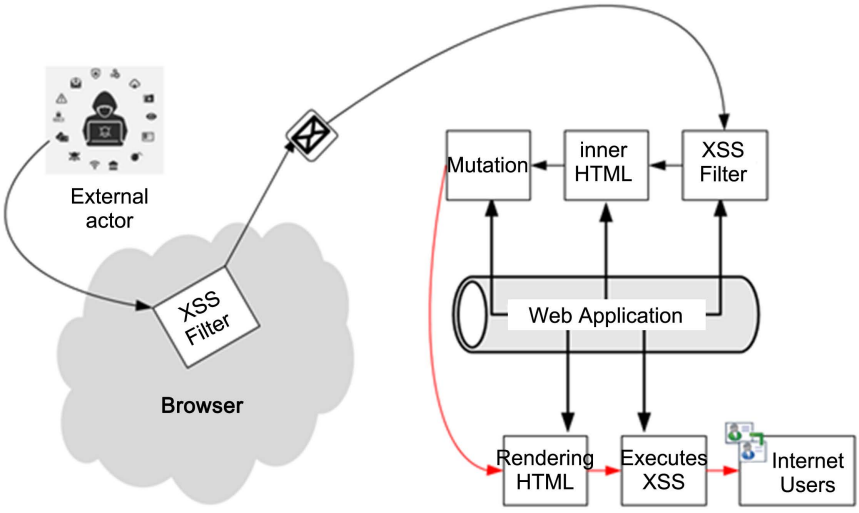


Figure 6. Mutation-based cross-site scripting (mXSS) attack scenario.

When an external actor injects something that appears safe, as shown in Figure 6, the browser rewrites and modifies it while processing the HTML, resulting in a mutated XSS attack [19]. This makes it incredibly difficult to find and sanitize bugs in application logic. Despite its novelty and widespread misinterpretation, mXSS attacks have been utilized to bypass the most sophisticated XSS filters available. mXSS has been used to circumvent solutions such as DOMPurify [20], OWASP AntiSamy, and Google Caja, and a large number of popular web apps (especially email clients) have been discovered to be vulnerable [21] [17]. At its foundation, mXSS works by employing filter-safe payloads that mutate into insecure payloads after filtration. All major browsers are vulnerable to mXSS attacks. Developers must understand how browsers handle optimizations and conditional expressions when rendering DOM nodes.

3. Composition of XSS Comparative Research Data Sources

This research utilizes a subset of the Global dataset containing CVE and CWE Security vulnerability database [22]. However, I concentrated only on the software development component of the information comprising CVE details for XSS vulnerability evaluation, as shown in Figure 7. The data consists of the vulnerability's CVE-ID, CWE-ID, Explanation, severity, and CVSS and the CWE names under which the vulnerability falls.

However, the abbreviation and acronyms used in this survey are carefully explained in Section 3.1.

As shown in Figure 7, these were the category of the dataset used from a programming perspective. The results from this survey were thoroughly analyzed to determine the annual trends in XSS vulnerability.

In [34]:
```
year_list=[]
for i in range(0,len(goodSD["CVE-ID"])):
    a=goodSD["CVE-ID"][i].split("-")[1]
    year_list.append(a)

goodSD.insert(loc=2, column='Year', value=year_list)
goodSD.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59833 entries, 0 to 59832
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   List         59833 non-null  int64
 1   CVE-ID       59833 non-null  object
 2   Year         59833 non-null  object
 3   CVSS-V3      59833 non-null  object
 4   CVSS-V2      59833 non-null  object
 5   SEVERITY     59833 non-null  object
 6   Explanation  59833 non-null  object
 7   CWE-ID       59833 non-null  object
 8   CWE-NAME     59833 non-null  object
dtypes: int64(1), object(8)
memory usage: 4.1+ MB
```

In [36]:
```
goodSD.describe(include=["object", "bool"])
```

Out[36]:

	CVE-ID	Year	CVSS-V3	CVSS-V2	SEVERITY	Explanation	CWE-ID	CWE-NAME
count	59833	59833	59833	59833.0	59833	59833	59833	59833
unique	59060	24	75	64.0	5	55840	150	152
top	CVE-2018-1000224	2020	None	4.3	MEDIUM	Multiple vulnerabilities in the web-based mana...	CWE-79	Improper Neutralization of Input During Web Pa...
freq	5	9557	13825	16503.0	28384	122	18566	18564

Figure 7. A brief overview of the dataset used for analyzing XSS vulnerability.

3.1. Abbreviations and Acronyms

XSS = Cross-Site Scripting;
DOM = Document Object Model;
mXSS = Mutation-Based Cross-Site Scripting;
NVD [23] [24] = National Vulnerability Database;
CVE [25] = Common Vulnerabilities and Exposures;
CWE [26] = Common Weakness Enumeration;
CVSS = Common Vulnerability Scoring System.

3.2. Comparative of the Top 20 Software Development Vulnerabilities

The pie charts below illustrate the number of the top 20 Software Development Vulnerabilities based on CWE Name from 2014 to 2022. Over the last nine years, the most frequent report of a cross-site scripting (XSS) vulnerability has been alarmingly received, as shown in Figure 8. I used python Jupiter Notebook [27] to analyze the data.

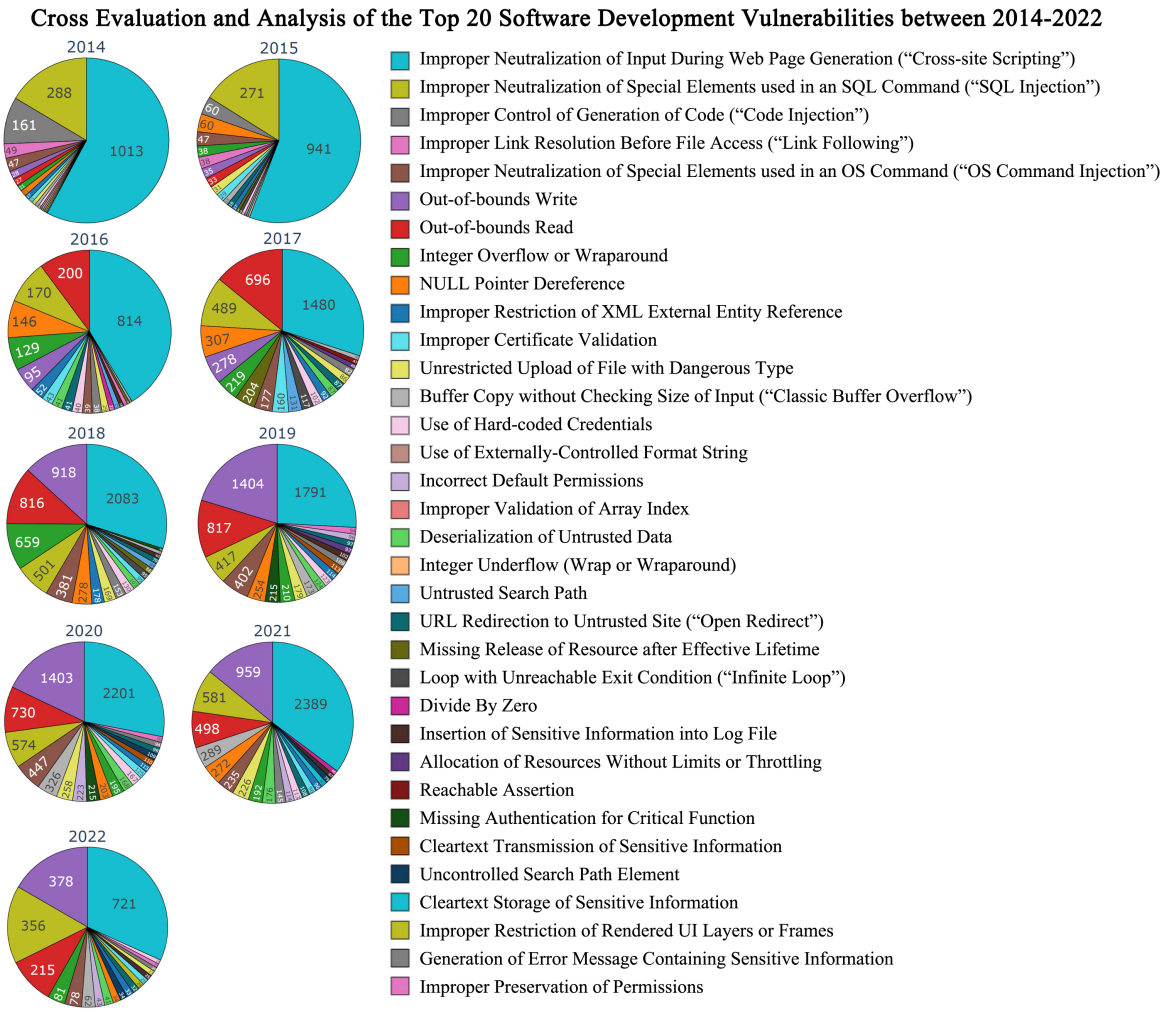


Figure 8. Comparative analysis of XSS vulnerability’s yearly trends.

4. Related Works

Different security organizations have revealed that XSS has been prevalent in internet security threats in the past years. Cross-Site Scripting (XSS) vulnerability has infiltrated approximately 70% [28] of web applications, including MySpace, Cisco, NASA, Facebook, Twitter, Google, YouTube, eBay, ads.tiktok.com [29], etc. Its emergence is primarily due to security flaws in web application development and incorrect input validation submitted by users in website input fields. The Samy MySpace worm in 2005 brought the XSS vulnerability to the notice of a wider audience globally [30]. So far, a wide variety of XSS attacks have been discussed. Interestingly, after conducting a comprehensive survey and reading over sixty research papers and publications, I have provided in this paper as Protective Approaches the defensive mechanisms revealed by previous researchers concerning XSS vulnerabilities. These defensive measures assist us in identifying and categorizing the articles based on the model employed to resolve the web application security problems.

5. XSS Prevention and Defense Mechanism

The XSS prevention and defense mechanism are explicitly explained in the following sections:

5.1. Preventive Measures and Standard Procedures for Cross-Site Scripting Attack

This section emphasizes most of the standard solutions that can be adopted to significantly reduce the impact of XSS attacks [31] [32]. It emphasizes on describing the XSS mitigation rules that developers can implement to prevent XSS attacks from occurring. It's evident that these techniques aren't magic; they're ineffectual without adequate awareness of users.

It is illustrated from **Figure 8** that just two vulnerabilities are dominating the field of web application security attacks, specifically XSS and SQL injection vulnerabilities. Developers can now use numerous preventive measures to keep themselves safe from XSS attacks. Data entered by the user that isn't trusted is protected using a combination of filtering, escape, and sanitization procedures. The following **Table 1** and **Table 2** describe each technique:

There are two varieties of escaping: input escaping and output escaping. Practical input escaping requires detecting the context of the untrusted data inserted correctly. In contrast, output escaping is performed to the response web page's written data. This also considers the data's context, which is essential for mitigating stored XSS attacks.

5.2. Researchers' Defensive Techniques for XSS Attacks (Advantages & Disadvantages)

The proliferation of XSS vulnerabilities attracts the interest of researchers and developers of security solutions. The variety of XSS attacks that each solution is

Table 1. General methods for preventing XSS attacks.

Technique	Explanation
Filtering	This implies that any unsafe user input must be filtered to remove dangerous phrases like the <script> tags, and event handlers in HTML that appear to be suspicious like onActivate() and onClick(), JavaScript workings, style sheet tags, etc. [33].
Escaping	Escape or encode to avoid XSS [34]. This prevents dangerous browser scripts from running. This signifies that the browser will simply store the data entered by the user without taking any further action with it.
HTML Entity Encoding	HTML [35] body tags are used to escape suspicious data like div, p, td, etc. I gave HTML entity escaping examples in Table 2.
Attribute Value Escaping	This prevents untrusted data from being directly entered into suspected attributes such as "href," "src," "style," and so on. Characters containing ASCII values below 256 are encoded using & #HH, where HH = hexadecimal value, leaving alphanumeric characters alone [36].
JavaScript Escaping	Script blocks and event handlers in JavaScript are more vulnerable to the XSS flaw. As a result, they use uxxxx, or Unicode escaping format, to process data entered using these methods, where x = integer [37].
URL Escaping	Because the untrusted data can only be found in parameter values, the encoding is applied to them. The escaping format is % HH [38].
CSS Escaping	For injection reasons, style sheets can also be used. As a result, this encoding employs the \HH and & \HHHH escaping formats [39].
Sanitization	It is another strategy for preventing an XSS attack. This guarantees that the data supplied matches the format anticipated for that particular input field on the website. HtmlSanitizer by OWASP, Ruby on Rails SanitizeHelper, DOMpurify, PHP HTML purifier, Python Bleach, and others do sanitization [40].
Content Security Policy	Mozilla suggested a security prototype called Content Security Policy (CSP) to address web application security vulnerabilities like XSS [41]. This permits a website developer to designate where to access external online resources.
Data Validation	This technique ensures that the submitted data adheres to the syntactical limitations set for that site, preventing unwanted and dangerous content. In different languages, such as PHP, functions such as is numeric(), preg match(), and others are defined to validate the data, or you can use regular expressions to validate the data technically [42].

Table 2. HTML entity encoding [43] [44] [45] [46].

Character	Encoded Format
/	/ or & #47
‘	‘ or & #39
“	& quot; or & #34
>	> or & #62
<	< or & #60
&	& amp; or & #38
#	& #35
)	& #41
(	& #40

designed to defend against has inspired the development of a wide variety of countermeasures. Based on the measures of their implementation model, I have grouped these solutions or techniques into five categories: client-side techniques, server-side techniques, machine learning techniques, and proxy-based techniques. In the following subsections, I have emphasized the most significant and effective methods proposed by the researchers as advantages and observed limitations to those approaches as disadvantages. In the appendix, you can find more information about the researchers' techniques.

6. Conclusions and Suggestions

This paper presents a comprehensive and in-depth survey on XSS attacks and the defensive techniques emphasized in the previous and current research literature. I have provided a global dataset combining CVE and CWE Security vulnerability information, taking into account the risk of XSS and how it is rapidly limiting the scientific endeavors of researchers worldwide. The author also offered a graphical representation of the annual trends of XSS attacks based on a comparative investigation of CWE names.

As indicated in Section 5.2, I have highlighted the impact of XSS attacks and the interminable effort given by the research community to combat XSS attacks. Along with its advantages and disadvantages, XSS defensive approaches to prior and recent efforts in various fields have been broadly classified. Existing XSS defensive techniques were separated into the following categories: machine learning technique, client-size technique, proxy-side technique, server-size technique, and combined technique. However, the vast majority of the cutting-edge XSS defensive techniques available in this paper protect against the more common types of XSS vulnerabilities, such as stored and reflected XSS. Presently, no dependable solution can provide appropriate protection against the recently found form of XSS attack known as DOM and mutation-based XSS attacks. These at-

tacks have been identified as a potential security risk. This study recommendation emphasizes the importance of developing solutions capable of offering effective defense against the newly identified variant of XSS. Using the survey results, we believe that the research community can better understand XSS protection measures and that this survey can also guide the development of more integrated and pragmatic security solutions. This survey suggested an efficient and robust XSS defensive architecture for future research. This study significantly contributes to the development of effective defensive mechanisms to limit the effects of such attacks on rapidly expanding web application platforms. Evaluation of existing XSS attack defensive solutions at the client-side, proxy-side, and server-side levels, as well as a machine learning technique that will undoubtedly aid in the evaluation of the impact of such an advanced level attack.

Combining static testing, dynamic testing, code auditing with secure coding, and ongoing initiatives to educate users about XSS developing vulnerabilities is critical. XSS will persist unless internet users become more aware of their security and privacy and software developers construct secure programs. According to this survey, XSS attacks can seize control of vital services and sensitive data if these safeguards are not established and maintained regularly.

Acknowledgements

This publication was made possible by the direction of the research laboratory of Hunan University's College of Computer Science and Electronic Engineering. I am grateful for the opportunity to utilize the facility and necessary electronic equipment to complete the data analysis task for this research.

I would like to express my gratitude to the entire research community for pointing me in the right direction and providing clarity regarding the principles that support web application security through the use of papers, books, surveys, online articles, and blogs.

Conflicts of Interest

The author states that there are no competing interests involved. This article's structure, as well as its contents and authorship, are solely the author's responsibility.

References

[1] Kirsten, S. (2016) Cross Site Scripting (XSS) Software Attack. https://owasp.org/www-community/attacks/xss/

[2] Agrawal, D.P. and Wang, H. (2018) Computer and Cyber Security. Auerbach Publications, New York. https://doi.org/10.1201/9780429424878

[3] Jiang, F., Fu, Y., Gupta, B.B., Liang, Y., Rho, S., Lou, F., *et al.* (2020) Deep Learning Based Multi-Channel Intelligent Attack Detection for Data Security. *IEEE Transactions on Sustainable Computing*, **5**, 204-212. https://doi.org/10.1109/TSUSC.2018.2793284

[4] Baş Seyyar, M., Çatak, F.Ö. and Gül, E. (2018) Detection of Attack-Targeted Scans from

the Apache HTTP Server Access Logs. *Applied Computing and Informatics*, **14**, 28-36. https://doi.org/10.1016/j.aci.2017.04.002

[5] Chen, H.-C., Nshimiyimana, A., Damarjati, C. and Chang, P.-H. (2021) Detection and Prevention of Cross-Site Scripting Attack with Combined Approaches. 2021 *International Conference on Electronics, Information, and Communication* (*ICEIC*), Jeju, 31 January-3 February 2021, 1-4. https://doi.org/10.1109/ICEIC51217.2021.9369796

[6] Gan, J.-M., Ling, H.-Y. and Leau, Y.-B. (2021) A Review on Detection of Cross-Site Scripting Attacks (XSS) in Web Security. *International Conference on Advances in Cyber Security*, Penang, 8-9 December 2020, 685-709. https://link.springer.com/chapter/10.1007/978-981-33-6835-4_45

[7] Wibowo, R.M. and Sulaksono, A. (2021) Web Vulnerability Through Cross Site Scripting (XSS) Detection with OWASP Security Shepherd. *Indonesian Journal of Information Systems*, **3**, 149-59. https://doi.org/10.24002/ijis.v3i2.4192

[8] Dora, J.R. and Nemoga, K. (2021) Ontology for Cross-Site-Scripting (XSS) Attack in Cybersecurity. *Journal of Cybersecurity and Privacy*, **2021**, 319-339. https://doi.org/10.3390/jcp1020018

[9] Nirmal, K., Janet, B. and Kumar, R. (2018) Web Application Vulnerabilities—The Hacker's Treasure. 2018 *International Conference on Inventive Research in Computing Applications* (*ICIRCA*), Coimbatore, 11-12 July 2018, 58-62. https://doi.org/10.1109/ICIRCA.2018.8597221

[10] Cui, Y., Cui, J. and Hu, J. (2020) A Survey on XSS Attack Detection and Prevention in Web Applications. *Proceedings of the* 2020 12*th International Conference on Machine Learning and Computing*, Shenzhen, 15-17 February 2020, 443-449. https://doi.org/10.1145/3383972.3384027

[11] Khazal, I. and Hussain, M. (2021) Server Side Method to Detect and Prevent Stored XSS Attack. *Iraqi Journal for Electrical and Electronic Engineering*, **17**, 58-65. https://doi.org/10.37917/ijeee.17.2.8

[12] Revyakina, Y., Cherckesova, L., Safaryan, O., Korochentsev, D., Boldyrikhin, N. and Ivanov, Y. (2020) Possibilities of Conducting XSS-Attacks and the Development of Countermeasures. *E3S Web of Conferences*, **224**, Article No. 01040. https://doi.org/10.1051/e3sconf/202022401040

[13] Maurel, H., Vidal, S. and Rezk, T. (2022) Statically Identifying XSS Using Deep Learning. *Science of Computer Programming*, **219**, Article ID: 102810. https://doi.org/10.1016/j.scico.2022.102810

[14] Hickling, J. (2021) What Is DOM XSS and Why Should You Care? *Computer Fraud & Security*, **2021**, 6-10. https://doi.org/10.1016/S1361-3723(21)00040-3

[15] Ninawe, S. and Wajgi, R. (2020) Detection of DOM-Based XSS Attack on Web Application. *Intelligent Communication Technologies and Virtual Mobile Networks* 2019, Tirunelveli, 14-15 February 2019, 633-641. https://link.springer.com/chapter/10.1007/978-3-030-28364-3_65

[16] Wang, P., Bangert, J. and Kern, C. (2021) If It's Not Secure, It Should Not Compile: Preventing DOM-Based XSS in Large-Scale Web Development with API Hardening. 2021 *IEEE/ACM* 43*rd International Conference on Software Engineering* (*ICSE*), Madrid, 22-30 May 2021, 1360-1372. https://doi.org/10.1109/ICSE43902.2021.00123

[17] Hoffman, A. (2020) Web Application Security: Exploitation and Countermeasures for Modern Web Applications. O'Reilly Media, Inc., Sebastopol. https://books.google.com/books?hl=en&lr=&id=3R3UDwAAQBAJ&oi=fnd&pg=PR2&dq=Web+application+security%E2%80%AF:+exploitation+and+countermeasu

res+for+modern+web+applications&ots=PGdlEp9ORy&sig=0EKCDxN_UPA9rpVKQvwPPjvzmyk#v=onepage&q=Web%20application%20security%E2%80%AF%3A%20exploitation%20and%20countermeasures%20for%20modern%20web%20applications&f=false

[18] Remya, S. and Praveen, K. (2016) Protecting the Augmented Browser Extension from Mutation Cross-Site Scripting. *Proceedings of the* 2*nd International Conference on Computer and Communication Technologies*, Vol. 1, Hyderabad, 24-26 July 2015, 215-223. https://link.springer.com/chapter/10.1007/978-81-322-2517-1_22

[19] Kaur, J. and Garg, U. (2021) A Detailed Survey on Recent XSS Web-Attacks Machine Learning Detection Techniques. 2021 2*nd Global Conference for Advancement in Technology* (*GCAT* 2021), Bangalore, 1-3 October 2021, 1-6. https://doi.org/10.1109/GCAT52182.2021.9587569

[20] Pazos, J.C., Légaré, J.S. and Beschastnikh, I. (2021) XSnare: Application-Specific Client-Side Cross-Site Scripting Protection. *Proceedings of* 2021 *IEEE International Conference on Software Analysis*, *Evolution and Reengineering* (*SANER* 2021), Honolulu, 9-12 March 2021, 154-165. https://doi.org/10.1109/SANER50967.2021.00023

[21] Mohammadi, M., Chu, B. and Richter Lipford, H. (2019) Automated Repair of Cross-Site Scripting Vulnerabilities through Unit Testing. *Proceedings of* 2019 *IEEE* 30*th International Symposium on Software Reliability Engineering Workshops*, (*ISSREW* 2019), Berlin, 27-30 October 2019, 370-377. https://doi.org/10.1109/ISSREW.2019.00098

[22] Wang, Y., Zhou, Y., Zou, X., Miao, Q. and Wang, W. (2020) The Analysis Method of Security Vulnerability Based on the Knowledge Graph. 2020 10*th International Conference on Communication and Network Security*, Tokyo, 27-29 November 2020, 135-145. https://doi.org/10.1145/3442520.3442535

[23] Williams, M.A., Dey, S., Barranco, R.C., Naim, S.M., Hossain, M.S. and Akbar, M. (2018) Analyzing Evolving Trends of Vulnerabilities in National Vulnerability Database. 2018 *IEEE International Conference on Big Data* (*Big Data*), Seattle, 10-13 December 2018, 3011-3020. https://doi.org/10.1109/BigData.2018.8622299

[24] Forain, I., de Oliveira Albuquerque, R. and de Sousa Júnior, R.T. (2022) Towards System Security: What a Comparison of National Vulnerability Databases Reveals. 2022 17*th Iberian Conference on Information Systems and Technologies* (*CISTI*), Madrid, 22-25 June 2022, 1-6. https://doi.org/10.23919/CISTI54924.2022.9820232

[25] Guo, H., Xing, Z., Chen, S., Li, X., Bai, Y. and Zhang, H. (2021) Key Aspects Augmentation of Vulnerability Description based on Multiple Security Databases. 2021 *IEEE* 45*th Annual Computers*, *Software*, *and Applications Conference* (*COMPSAC*), Madrid, 12-16 July 2021, 1020-1025. https://doi.org/10.1109/COMPSAC51774.2021.00138

[26] Wang, T., Qin, S. and Chow, K.P. (2021) Towards Vulnerability Types Classification Using Pure Self-Attention: A Common Weakness Enumeration Based Approach. 2021 *IEEE* 24*th International Conference on Computational Science and Engineering* (*CSE*), Shenyang, 20-22 October 2021, 146-153. https://doi.org/10.1109/CSE53436.2021.00030

[27] Mantha, B.R.K., Jung, Y. and Garcia De Soto, B. (2020) Implementation of the Common Vulnerability Scoring System to Assess the Cyber Vulnerability in Construction Projects. *Creative Construction E-Conference* 2020, Online, 28 June-1 July 2020, 117-124.

[28] Fangohr, H., Kluyver, T. and DiPierro, M. (2021) Jupyter in Computational Science.

Computing in Science & Engineering, **23**, 5-6.
https://doi.org/10.1109/MCSE.2021.3059494

[29] Pradeepa, P.K. (2022) A Survey on an Investigation of Detection & Prevention Methods for Cross-Site Scripting (XSS) Attacks. *International Journal of Advanced Research in Science*, *Communication and Technology*, 405-413.

[30] Kaur, J. and Garg, U. (2021) A Detailed Survey on Recent XSS Web-Attacks Machine Learning Detection Techniques. 2021 2*nd Global Conference for Advancement in Technology* (*GCAT* 2021), Bangalore, 1-3 October 2021, 1-6.
https://doi.org/10.1109/GCAT52182.2021.9587569

[31] Sahoo, S.R. and Gupta, B.B. (2019) Classification of Various Attacks and Their Defence Mechanism in Online Social Networks: A Survey. *Enterprise Information Systems*, **13**, 832-864. https://doi.org/10.1080/17517575.2019.1605542

[32] Kaur, G., Pande, B., Bhardwaj, A., Bhagat, G. and Gupta, S. (2018) Efficient Yet Robust Elimination of XSS Attack Vectors from HTML5 Web Applications Hosted on OSN-Based Cloud Platforms. *Procedia Computer Science*, **125**, 669-675.
https://doi.org/10.1016/j.procs.2017.12.086

[33] Xu, G., Xie, X., Huang, S., Zhang, J., Pan, L., Lou, W., *et al.* (2020) JSCSP: A Novel Policy-Based XSS Defense Mechanism for Browsers. *IEEE Transactions on Dependable and Secure Computing*, **19**, 826-878.
https://doi.org/10.1109/TDSC.2020.3009472

[34] Lala, S.K., Kumar, A. and Subbulakshmi, T. (2021) Secure Web Development Using OWASP Guidelines. *Proceedings of* 5*th International Conference on Intelligent Computing and Control Systems* (*ICICCS* 2021), Madurai, 6-8 May 2021, 323-332.
https://doi.org/10.1109/ICICCS51141.2021.9432179

[35] Sahin, M., Ünlü, T., Hébert, C., Shepherd, L.A., Coull, N. and Lean, C.M. (2022) Measuring Developers' Web Security Awareness from Attack and Defense Perspectives. 2022 *IEEE Security and Privacy Workshops* (*SPW*), San Francisco, 22-26 May 2022, 31-43.

[36] Gupta, S. and Gupta, B.B. (2016) XSS-SAFE: A Server-Side Approach to Detect and Mitigate Cross-Site Scripting (XSS) Attacks in JavaScript Code. *Arabian Journal for Science and Engineering*, **41**, 897-920. https://doi.org/10.1007/s13369-015-1891-7

[37] Gupta, S. and Gupta, B.B. (2018) XSS-Secure as a Service for the Platforms of Online Social Network-Based Multimedia Web Applications in Cloud. *Multimedia Tools and Applications*, **77**, 4829-4861.
https://link.springer.com/article/10.1007/s11042-016-3735-1

[38] Gupta, B.B., Gupta, S. and Chaudhary, P. (2017) Enhancing the Browser-Side Context-Aware Sanitization of Suspicious HTML5 Code for Halting the DOM-Based XSS Vulnerabilities in Cloud. *International Journal of Cloud Applications and Computing*, **7**, 1-31. https://doi.org/10.4018/IJCAC.2017010101

[39] Caliwag, J.A., Pagaduan, R.A., Castillo, R.E. and Ramos, W.V.J. (2019) Integrating the Escaping Technique in Preventing Cross Site Scripting in an Online Inventory System. *Proceedings of the* 2019 2*nd International Conference on Information Science and Systems*, Tokyo, 16-19 March, 110-114.
https://doi.org/10.1145/3322645.3322696

[40] Stritter, B., Freiling, F., König, H., Rietz, R., Ullrich, S., *et al.* (2016) Cleaning up Web 2.0's Security Mess-At Least Partly. *IEEE Security and Privacy*, **14**, 48-57.
https://doi.org/10.1109/MSP.2016.31

[41] Singh, N., Meherhomji, V. and Chandavarkar, B.R. (2020) Automated versus Manual Approach of Web Application Penetration Testing. 2020 11*th International Confe-*

rence on Computing, Communication and Networking Technologies (*ICCCNT*), Kharagpur, 1-3 July 2020, 1-6. https://doi.org/10.1109/ICCCNT49239.2020.9225385

[42] Calzavara, S., Rabitti, A. and Bugliesi, M. (2018) Semantics-Based Analysis of Content Security Policy Deployment. *ACM Transactions on the Web*, **12**, Article No, 10. https://doi.org/10.1145/3149408

[43] Breck, E., Polyzotis, N., Roy, S., Whang, S.E. and Zinkevich, M. (2019) Data Validation for Machine Learning. https://proceedings.mlsys.org/book/2019/file/5878a7ab84fb43402106c575658472fa-Paper.pdf

[44] Serrano, M. and Prunet, V. (2016) A Glimpse of Hopjs. *ACM SIGPLAN Notices*, **51**, 180-192. https://doi.org/10.1145/3022670.2951916

[45] Rose, F., Toher, C., Gossett, E., Oses, C., Nardelli, M.B., Fornari, M., *et al.* (2017) AFLUX: The LUX Materials Search API for the AFLOW Data Repositories. *Computational Materials Science*, **137**, 362-370. https://doi.org/10.1016/j.commatsci.2017.04.036

[46] Argyros, G., Stais, I., Kiayias, A. and Keromytis, A.D. (2016) Back in Black: Towards Formal, Black Box Analysis of Sanitizers and Filters. 2016 *IEEE Symposium on Security and Privacy* (*SP*), San Jose, 22-26 May 2016, 91-109. https://ieeexplore.ieee.org/abstract/document/7546497

[47] Pham, T.T.T., Hoang, V.N. and Ha, T.N. (2018) Exploring Efficiency of Character-Level Convolution Neuron Network and Long Short Term Memory on Malicious URL Detection. *Proceedings of the* 2018 *VII International Conference on Network, Communication and Computing*, Taipei, 14-16 December 2018, 82-86. https://doi.org/10.1145/3301326.3301336

[48] Zhang, Q., Chen, H. and Sun, J. (2010) An Execution-Flow Based Method for Detecting Cross-Site Scripting Attacks. *The* 2*nd International Conference on Software Engineering and Data Mining*, Chengdu, 23-25 June 2010, 160-165. https://ieeexplore.ieee.org/abstract/document/5542934

[49] Scholte, T., Robertson, W., Balzarotti, D. and Kirda, E. (2012) Preventing Input Validation Vulnerabilities Inweb Applications through Automated Type Analysis. *Proceedings of International Computer Software and Applications Conference*, Izmir, 16-20 July 2012, 233-243. https://doi.org/10.1109/COMPSAC.2012.34

[50] Xiao, W., Sun, J., Chen, H. and Xu, X. (2014) Preventing Client Side XSS with Rewrite Based Dynamic Information Flow. *Proceedings of International Symposium on Parallel Architectures, Algorithms and Programming* (*PAAP*), Beijing, 13-15 July 2014, 238-243. https://doi.org/10.1109/PAAP.2014.10

[51] Stock, B., Pfistner, S., Kaiser, B., Lekies, S. and Johns, M. (2015) From Facepalm to Brain Bender: Exploring Client-Side Cross-Site Scripting. *Proceedings of the* 22*nd ACM SIGSAC Conference on Computer and Communications Security*, Denver, 12-16 October 2015, 1419-1430. https://doi.org/10.1145/2810103.2813625

[52] Parameshwaran, I., Budianto, E., Shinde, S., Dang, H., Sadhu, A. and Saxena, P. (2015) DexterJS: Robust Testing Platform for DOM-Based XSS Vulnerabilities. *Proceedings of the* 2015 10*th Joint Meeting on Foundations of Software Engineering*, Bergamo, 30 August-4 September 2015, 946-949. https://doi.org/10.1145/2786805.2803191

[53] Usha, G., Kannimuthu, S., Mahendiran, P.D., Shanker, A.K. and Venugopal, D. (2020) Static Analysis Method for Detecting Cross Site Scripting Vulnerabilities. *International Journal of Information and Computer Security*, **13**, 32-47. https://doi.org/10.1504/IJICS.2020.108123

[54] Wang, R., Jia, X., Li, Q. and Zhang, D. (2015) Improved N-Gram Approach for Cross-Site Scripting Detection in Online Social Network. 2015 *Science and Information Conference* (*SAI*), London, 28-30 July 2015, 1206-1212. https://doi.org/10.1109/SAI.2015.7237298

[55] Mokbal, F.M.M., Wang, D., Imran, A., Jiuchuan, L., Akhtar, F. and Wang, X. (2019) MLPXSS: An Integrated XSS-Based Attack Detection Scheme in Web Applications Using Multilayer Perceptron Technique. *IEEE Access*, **7**, 100567-100580. https://doi.org/10.1109/ACCESS.2019.2927417

[56] Moniruzzaman, M., Bagirov, A., Gondal, I. and Brown, S. (2018) A Server Side Solution for Detecting WebInject: A Machine Learning Approach. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Melbourne, 3 June 2018, 162-167. https://doi.org/10.1007/978-3-030-04503-6_16

[57] Tariq, I., Sindhu, M.A., Abbasi, R.A., Khattak, A.S., Maqbool, O. and Siddiqui, G.F. (2021) Resolving Cross-Site Scripting Attacks through Genetic Algorithm and Reinforcement Learning. *Expert Systems with Applications*, **168**, Article ID: 114386. https://doi.org/10.1016/j.eswa.2020.114386

[58] Fang, Y., Huang, C., Xu, Y. and Li, Y. (2019) RLXSS: Optimizing XSS Detection Model to Defend Against Adversarial Attacks Based on Reinforcement Learning. *Future Internet*, **11**, Article 177. https://doi.org/10.3390/fi11080177

[59] Fang, Y., Li, Y., Liu, L. and Huang, C. (2018) DeepXSS: Cross Site Scripting Detection Based on Deep Learning. *Proceedings of the* 2018 *International Conference on Computing and Artificial Intelligence*, Chengdu, 12-14 March 2018, 47-51. https://doi.org/10.1145/3194452.3194469

[60] Kaur, G., Malik, Y., Samuel, H. and Jaafar, F. (2018) Detecting Blind Cross-Site Scripting Attacks Using Machine Learning. *Proceedings of the* 2018 *International Conference on Signal Processing and Machine Learning*, Shanghai, 28-30 November 2018, 22-25. https://doi.org/10.1145/3297067.3297096

[61] Lekies, S., Stock, B. and Johns, M. (2013) 25 Million Flows Later: Large-Scale Detection of DOM-Based XSS. *Proceedings of the* 2013 *ACM SIGSAC Conference on Computer & Communications Security*, Berlin, 4-8 November 2013, 1193-1204. https://doi.org/10.1145/2508859.2516703

[62] Van Acker, S., Nikiforakis, N., Desmet, L., Joosen, W. and Piessens, F. (2012) FlashOver: Automated Discovery of Cross-Site Scripting Vulnerabilities in Rich Internet Applications. *Proceedings of the* 7*th ACM Symposium on Information, Computer and Communications Security*, Seoul, 2-4 May 2012, 12-13. https://doi.org/10.1145/2414456.2414462

[63] Vishnu, B.A. and Jevitha, K.P. (2014) Prediction of Cross-Site Scripting Attack Using Machine Learning Algorithms. *Proceedings of the* 2014 *International Conference on Interdisciplinary Advances in Applied Computing*, Amritapuri, October 2014, Article No. 55. https://doi.org/10.1145/2660859.2660969

[64] Rocha, T.S. and Souto, E. (2014) ETSSDetector: A Tool to Automatically Detect Cross-Site Scripting Vulnerabilities. 2014 *IEEE* 13*th International Symposium on Network Computing and Applications*, Cambridge, 21-23 August 2014, 306-309. https://doi.org/10.1109/NCA.2014.53

[65] Khan, N., Abdullah, J. and Khan, A.S. (2015) Towards Vulnerability Prevention Model for Web Browser Using Interceptor Approach. 2015 9*th International Conference on IT in Asia* (*CITA*), Sarawak, 4-5 August 2015, 1-5. https://doi.org/10.1109/CITA.2015.7349842

[66] Ruse, M.E. and Basu, S. (2013) Detecting Cross-Site Scripting Vulnerability Using

Concolic Testing. 2013 10*th International Conference on Information Technology*: *New Generations*, Las Vegas, 15-17 April 2013, 633-638. https://doi.org/10.1109/ITNG.2013.97

[67] Dong, G., Zhang, Y., Wang, X., Wang, P. and Liu, L. (2014) Detecting Cross Site Scripting Vulnerabilities Introduced by HTML5. 2014 11*th International Joint Conference on Computer Science and Software Engineering* (JCSSE), Chon Buri, 14-16 May 2014, 319-323. https://doi.org/10.1109/JCSSE.2014.6841888

[68] Gupta, M.K., Govil, M.C., Singh, G. and Sharma, P. (2015) XSSDM: Towards Detection and Mitigation of Cross-Site Scripting Vulnerabilities in Web Applications. 2015 *International Conference on Advances in Computing*, *Communications and Informatics* (*ICACCI*), Kochi, 10-13 August 2015, 2010-2015. https://doi.org/10.1109/ICACCI.2015.7275912

[69] Duchene, F., Rawat, S., Richier, J.-L. and Groz, R. (2014) KameleonFuzz: Evolutionary Fuzzing for Black-Box XSS Detection. *Proceedings of the* 4*th ACM Conference on Data and Application Security and Privacy*, Association for Computing Machinery, San Antonio, March 2014, 37-48. https://doi.org/10.1145/2557547.2557550

[70] Lalia, S. and Sarah, A. (2018) XSS Attack Detection Approach Based on Scripts Features Analysis. 2018 *World Conference on Information Systems and Technologies*, Naples, 27-29 March 2018, 197-207. https://link.springer.com/chapter/10.1007/978-3-319-77712-2_19

[71] Steinhauser, A. and Tůma, P. (2019) DjangoChecker: Applying Extended Taint Tracking and Server Side Parsing for Detection of Context-Sensitive XSS Flaws. *Software*: *Practice and Experience*, **49**, 130-148. https://doi.org/10.1002/spe.2649

[72] Gupta, S. and Gupta, B.B. (2018) A Robust Server-Side JavaScript Feature Injection-Based Design for JSP Web Applications Against XSS Vulnerabilities. In: Bokhari, M.U., Agrawal, N. and Saini, D., Eds., *Cyber Security*, Springer, Singapore, 459-465. https://link.springer.com/chapter/10.1007/978-981-10-8536-9_43

[73] Gupta, S., Gupta, B.B. and Chaudhary, P. (2018) Hunting for DOM-Based XSS Vulnerabilities in Mobile Cloud-Based Online Social Network. *Future Generation Computer Systems*, **79**, 319-336. https://doi.org/10.1016/j.future.2017.05.038

[74] Nadji, Y., Saxena, P. and Song, D. (2009) Document Structure Integrity: A Robust Basis for Cross-Site Scripting Defense. National Down Syndrome Society, New York, 20. http://webblaze.cs.berkeley.edu/papers/nadji-saxena-song.pdf

[75] Panja, B., Gennarelli, T. and Meharia, P. (2015) 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. 2015 1*st Conference on Mobile and Secure Services* (*MOBISECSERV*), Gainesville, 20-21 February 2015, 1-7. https://ieeexplore.ieee.org/abstract/document/7072878

[76] Chaudhary, P., Gupta, B.B. and Gupta, S. (2018) Defending the OSN-Based Web Applications from XSS Attacks Using Dynamic JavaScript Code and Content Isolation. In: Kapur, P.K., Kumar, U. and Verma, A.K., Eds., *Quality*, *IT and Business Operations*: *Modeling and Optimization*, Springer, Singapore, 107-119. https://doi.org/10.1007/978-981-10-5577-5_9

[77] Gupta, S., Gupta, B.B. and Chaudhary, P. (2019) A Client-Server JavaScript Code Rewriting-Based Framework to Detect the XSS Worms from Online Social Network. *Concurrency and Computation*: *Practice and Experience*, **31**, Article No. e4646. https://doi.org/10.1002/cpe.4646

Appendix

A. Proxy-Based XSS prevention techniques

Table A1. Advantages and disadvantages of proxy-based XSS defensive techniques.

Advantages	Disadvantages
[47] presents a research paper in which the authors proposed that XSS attacks can be identified by investigating the implementation flow of an AJAX application. The JavaScript code is inspected on the browser side at the very beginning in order to generate a finite state machine for the typical mode of operation of the applications. The scripts that are encoded in the response web page are then monitored by this machine in conjunction with a proxy. Malicious flow and XSS attacks can take place if the machine's execution path does not match the machine's finite state machine.	This method is ineffective against XSS attacks based on the Document Object Model.
IPAAS is an input validation approach proposed by the authors [48]. After first interfering with the page containing the answer and retrieving all of the parameters, it then analyzes the context of those parameters. This leads to the development of input validation policies, which are subsequently applied to every page of reappearance on the internet for examination. If the requirements are not satisfied, the request will be denied; in any other case, it will not be. Failures in type learning are possible when bespoke query string formats are used.	The IPAAS parameter extractor may be unable to parse parameter key-value pairs in this approach.
The authors [49] proposed that this strategy employs the use of dynamic analysis of JavaScript code that is embedded within a web page. By using this method, an abstract syntax tree will be constructed for the internal representation of JavaScript code. Following that, the tree is transferred to the taint engine, which analyzes the JS code to determine whether or not it attempts to access the private data.	Performing its functions, however, incurs significant performance overhead and requires a substantial amount of processing time.
Essentially, the researchers [50] proposed that the objective of this approach is to detect any questionable JavaScript code. Tainted-browsing technology is used. A set of metrics is then established to help measure the impact of each attacking flow on the system.	This method cannot detect suspicious flow, for example, flows dependent on certain specified conditions, such as the value of a parameter in a URL.
The researchers [51] proposed that DOM-based XSS attack can no longer be carried out using this technique, thanks to its robustness. Taint tracking and exploit reporting are the foundations of this strategy. To a large extent, it gets rid of the JavaScript code that can't be trusted and then examines it on its own, following its execution flow, to determine whether or not it has been contaminated. It generates XSS test payloads based on the log information it receives.	Exploits are reported to the client after all vulnerabilities have been discovered. In terms of security, it does not guard against non-scripting code and has a negative impact on performance.

B. Machine learning XSS prevention techniques

Table A2. Advantages and disadvantages of machine learning XSS defensive techniques.

Advantages	Disadvantages
The researchers [52] proposed a data mining and static analysis approach for eliminating XSS vulnerabilities. The approach seeks to discover and eliminate harmful links from the source code. Their technique outperforms the upgraded ngram model. Following a discussion of the subclasses of XSS attacks, the paper briefly addresses the risks and concerns posed by XSS.	This approach cannot adequately prevent XSS against mXSS and DOM-based cross-site scripting (XSS) attacks.

Continued

The authors [53] proposed combining the machine-learning technique of classifiers with an upgraded n-gram approach to protect the social networking platform from XSS attacks.	If characteristics and examples are insufficient, it is possible that malicious pages won't be recognized, which will make the training effort for this strategy difficult.
The researchers [54] proposed a method for preventing cross-site scripting that utilizes ANN-Multilayer Perceptron in conjunction with dynamic feature extraction. When compared to other machine-learning algorithms, this strategy outperforms others.	For XSS assaults, it has not been tested on actual web applications that are used in the world today.
In [55] web page content can be distinguished from injected data using a technique proposed by the authors. This machine-learning-based approach is exclusive to banking websites. The model is trained using data from the DOM tree.	This approach takes more time since it involves removing features from the web page before sending it back to the server where it originated.
The researchers [56] Proposed a hybrid solution for preventing XSS in web applications. They claim that their method is the first of its kind since it blends a metaheuristic algorithm (the Genetic Algorithm) with a framework for machine learning. This combination distinguishes their methodology. They used a threat intelligence model and reinforcement learning in addition to GA and statistical inference to protect them from XSS attacks.	This strategy has not been put through any kind of proof-of-concept testing on real-world mission-critical web applications.
The authors [57] presented RLXSS, a method for detecting cross-site scripting attacks dependent on reinforcement learning, and uses both adversarial and retraining models. This method made use of XSS detection technologies like SafeDog and XSSChop in addition to DDQN (dueling deep Q networks), an escape technique, and a reward mechanism. The adversarial samples that were obtained from the adversarial model were included in the retraining model so that optimization could be performed on them.	This approach cannot work against mXSS attack that usually employs filter-safe payloads and mutate them into insecure payloads after filtration.
The authors [58] proposed a deep learning approach to the Cross-site scripting identification in which the original data is first decoded, and then the word2vec algorithm is used to acquire information regarding the qualities of XSS payloads. The input is then placed into a Model of the LSTM neural network. Cross-validation of the tenfold test is utilized in the last step of this analysis to see how well the proposed method compares to the ADTree and AdaBoost methods.	This approach is ineffective against DOM-based XSS attacks.
The authors [59] proposed a supervised machine learning method for detecting potentially hazardous links before they execute on the victim's computer. Their solution makes use of a Linear Support Vector Machine classifier to detect blind XSS attacks and differentiate between the primary characteristics of reflected and stored XSS attacks. JavaScript events were run during the features extraction process, which attackers use to inject malicious payloads. For testing purposes, a linearly separable dataset was used. Mutillidae, a free vulnerable website, was used to mimic a blind XSS attack.	This approach is entirely limited to handling DOM-base and mXSS attacks.
The authors [60] proposed a model for the detection of XSS that makes use of a metaheuristic approach known as a Genetic Algorithm.	This approach has not been tested on real-world, mission-critical web applications.

C. Client-side XSS prevention techniques

Table A3. Advantages and disadvantages of client-side XSS defensive techniques.

Advantages	Disadvantages
The researchers [61] have proposed a method for detecting DOM-based XSS attacks that employ dynamic taint tracking and context-sensitive sanitization.	This method is ineffective against stored XSS attacks.
The authors proposed that this [62] method is intended to minimize XSS attacks when used with Adobe Flash. This method also uses static analysis to detect suspicious input fields and dynamic analysis to test the suspect areas.	If the testing payload is executed, it leaves the system open to XSS attacks. When it comes to detecting XSS vulnerability sources, static analysis is only effective in a limited number of cases. Furthermore, it is only effective against malicious JavaScript code.
The researchers [63] proposed Machine-learning classifiers in the process. The set of data is then used in training classifiers to recognize XSS attacks once it has been extracted, examined, and prepared by taking the value of the URL parameter and the value of the JavaScript.	There is no automatic updating of a prepared dataset. As a result, a new attacking payload may be bypassed.
The researchers [64] proposed a method that operates by imitating the browser's behavior. It interacts with the website in issue and detects any potentially risky places before injecting a payload for testing the system's level of security. If the code executes, It is vulnerable to XSS attacks.	This approach cannot identify DOM-based XSS attacks.
The researchers [65] proposed a technique that operates as an intermediary between the client and the server who acts as an interceptor during the processing of a web page to detect the injection of malicious code. This method differentiates between static and dynamic websites. Vulnerabilities can be identified by injecting an attack payload into dynamic web pages. XSS attacks are possible in the event that the content is shown on the page.	This method is unable to identify DOM-based XSS attacks.

D. Server-side XSS prevention techniques

Table A4. Advantages and disadvantages of server-side XSS defensive techniques.

Advantages	Disadvantages
This method [66], according to the researchers, is intended for JSP-based web-related applications and is a jCute concolic testing. They employ static analysis and real-time monitoring. When an XSS attack is attempted, it helps to establish the relationship between input and output values that facilitate the attack.	Since this method relies on jCute concolic testing, output variables with more than three of the characters cannot be recognized.
The researchers [67] proposed that in addition to being able to detect XSS attack vectors constructed utilizing new HTML5 features, this approach is targeted for webmail applications. Five injection points in the webmail system are used to inject attack vectors for the purpose of testing. As the last step, it is determined whether or not an attack vector was thoroughly sanitized.	In this method, HTML5 tags and attributes are the sole attack vectors it considers, ignoring other potentially dangerous circumstances.
In [68], the authors have taken precautionary measures against XSS attacks by employing static analysis, pattern matching, and context-aware sanitization techniques.	In order to use this method, sanitized code must be manually entered into the website.

Continued

The researchers [69] who conducted the study hypothesized that fuzz testing activates XSS vulnerabilities. Fuzz testing is a black-box detection method that makes use of malicious payload injection into web applications. It's more accurate to think of it as a two-step extension of the LigRE model: first, the production of malicious input, and then the taint analysis in order to find the vulnerability. For instance, it avoids a cross-site scripting attack that is stored as well as reflected.	This would necessitate an application reset for live applications, which is not an option. Additionally, human interpretation is essential to the process of developing attack vectors.
In [70] script characteristics can be used to detect malicious script injection, according to the authors. These features are taken and then evaluated to see how they are used to create harmful scripts in this case. Once the malicious script and benign script are detected, they can be utilized to identify an XSS attack and prevent further damage.	Partially injected scripts and obfuscated script injection are ineffective with this strategy.
In [71] Django Checker is a dynamic taint analysis tool proposed by the authors. This method determines whether the primitives of the sanitizers that are already in use in the web application are proper. It also determines the context in which these attributes are used and assesses the appropriateness of implementing sanitization. It determines whether or not sanitization is context-sensitive.	This technique is limited to Django-based web applications and cannot detect DOM-based XSS attacks.
Researchers [72] have proposed a method based on discovering the discrepancies between inserted values and previously established values. Each site extracts JS code and tests to see if it differs from the known value. As a result, code injection flaws like XSS can be detected more easily.	However, if the Javascript context is ignored, XSS can also take advantage of other contexts, such as URL parameters and style sheet features. Attack vectors such as these can't be stopped by this method.

E. Combined XSS prevention techniques

Table A5. Advantages and disadvantages of combined XSS defensive techniques.

Advantages	Disadvantages
The researchers [73] have presented defensive strategies against DOM-based XSS attacks. Under normal circumstances, the DOM tree is constructed, scripting nodes are extracted, and a whitelist is created for future use. The DOM tree is generated for malicious websites, and the nodes of the DOM tree are parsed for injected script code. Any differences found between the whitelist and the XSS attack are viewed as suspicious compared to each other.	This technique may block the execution of harmless JavaScript code if the whitelist is not matched.
The researchers [74] proposed the usage of a client-server model to ensure the integrity of the document structure. This approach uses combined runtime tracking and randomization to prevent XSS attacks. As a result of this method, harmful data cannot affect web application content by manipulating the document structure.	This method, which needs modifications on both the client and the server, is unusable in preventing a DOM-based XSS attack.
As shown in [75] to identify and mitigate Cross-site Scripting (XSS) vulnerabilities on mobile browsers, the authors presented a method that is known as Buffer Based Cache Check. By utilizing a cache, you can avoid the time-consuming and resource-intensive process of continually transmitting the script whitelist to the web page. Instead, the server saves confirmed scripts that correspond to the last time the web page was browsed. If any deviations are discovered, it suggests suspicious activities such as XSS.	Code modifications on both the client and server sides are required for this method, which leads to a decrease in overall performance.

Continued

The researchers [76] have proposed a new approach to data cleaning using context-sensitive sanitization. Here, the server-side and client-side contexts are determined statically and dynamically. After this, sanitizers' primitives are applied to the vulnerable variable in accordance with its context.	This technique does not defend against malicious script code obtained from a third party.
The researchers [77] have proposed a client-server approach that extracts JavaScript code and analyzes it on the client-side. After decoding JS, the injected values are eventually matched with the suspicious variable contexts. As recommended by the authors, the presence of a match indicates an XSS assault.	The matching between requesting parameters and response parameters used in this technique is not capable of detecting DOM-based XSS attacks, which are client-side vulnerabilities.