Computer Science Review 52 (2024) 100634

Contents lists available at ScienceDirect

Computer Science Review

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

Review article

Twenty-two years since revealing cross-site scripting attacks: A systematic mapping and a comprehensive survey

Abdelhakim Hannousse [a,b,*], Salima Yahiouche [c], Mohamed Cherif Nait-Hamoud [a]

[a] *Laboratory of Vision and Artificial Intelligence (LAVIA), Larbi Tebessi University, BP 289, Tebessa, 12000, Algeria*
[b] *PI:MIS Laboratory, Université 8 Mai 1945 Guelma, BP 401, Guelma, 24000, Algeria*
[c] *LRS Laboratory, Badji Mokhtar University, BP 12, Annaba, 23000, Algeria*

ARTICLE INFO

Keywords:
Web security
Cross-site scripting
XSS attacks
XSS vulnerabilities
Systematic mapping
Survey

ABSTRACT

Cross-site scripting (XSS) is one of the major threats menacing the privacy of data and the navigation of trusted web applications. Since its disclosure in late 1999 by Microsoft security engineers, several techniques have been developed with the aim of securing web navigation and protecting web applications against XSS attacks. XSS has been and is still in the top 10 list of web vulnerabilities reported by the Open Web Applications Security Project (OWASP). Consequently, handling XSS attacks has become one of the major concerns of several web security communities. Despite the numerous studies that have been conducted to combat XSS attacks, the attacks continue to rise. This motivates the study of how the interest in XSS attacks has evolved over the years, what has already been achieved to prevent these attacks, and what is missing to restrain their prevalence. In this paper, we conduct a systematic mapping and a comprehensive survey with the aim of answering all these questions. We summarize and categorize existing endeavors that aim to handle XSS attacks and develop XSS-free web applications. The systematic mapping yielded 157 high-quality published studies. By thoroughly analyzing those studies, a comprehensive taxonomy is drawn out outlining various techniques used to prevent, detect, protect, and defend against XSS attacks and vulnerabilities. The study of the literature revealed a remarkable interest bias toward basic (84.71%) and JavaScript (81.63%) XSS attacks as well as a dearth of vulnerability repair mechanisms and tools (only 1.48%). Notably, existing vulnerability detection techniques focus solely on single-page detection, overlooking flaws that may span across multiple pages. Furthermore, the study brought to the forefront the limitations and challenges of existing attack detection and defense techniques concerning machine learning and content-security policies. Consequently, we strongly advocate the development of more suitable detection and defense techniques, along with an increased focus on addressing XSS vulnerabilities through effective detection (hybrid solutions) and repair strategies. Additionally, there is a pressing need for more high-quality studies to overcome the limitations of promising approaches such as machine learning and content-security policies while also addressing diverse XSS attacks in different languages. Hopefully, this study can serve as guidance for both the academic and practitioner communities in the development of XSS-free web applications.

1. Introduction

The use of web applications through the Internet has become an indispensable mean for different business and governmental organizations to reduce costs, speed up activities, improve the quality of services and reach as many targeted people as possible [1]. Users also get immense benefits from online provided services. However, those gains are not without risks; web applications requiring users' registrations through input forms are preferred targets for different hacking attacks, putting their own and users' confidential data at risks [2].

Cross-site scripting (XSS) is recognized as one of the most dangerous attacks threatening the security of several types of web applications since 1999 [3]. XSS attacks are code injection based attacks; attackers can get benefits from vulnerabilities, also named XSS, discovered in trusted web applications, their related plugins or hosting servers, to inject malicious scripts in the aim to compromise them. This way, explored pages of compromised web applications on users' browsers enable attackers to elevate their access-privileges, reveal sensitive user information retained by browsers such as cookies, usernames

[*] Corresponding author at: PI:MIS Laboratory, Université 8 Mai 1945 Guelma, BP 401, Guelma, 24000, Algeria.
E-mail addresses: hannousse.abdelhakim@univ-guelma.dz (A. Hannousse), salima.yahiouche@univ-tebessa.dz (S. Yahiouche), mohamed-cherif.nait-hamoud@univ-tebessa.dz (M.C. Nait-Hamoud).

https://doi.org/10.1016/j.cosrev.2024.100634
Received 31 August 2022; Received in revised form 31 July 2023; Accepted 17 April 2024
Available online 23 April 2024

and passwords on behalf of victim users. Practically, XSS attacks can be launched using various means, a typical scenario involves the insertion of malicious scripts into vulnerable web application input fields, often in the form of URL parameters. Target users are then enticed to click on manipulated links containing these scripts through social engineering techniques. Upon accessing such links, the malicious scripts are executed, enabling unauthorized access to sensitive user data stored in browsers or the stealthy upload of dangerous malware onto the target user's computer. Another common tactic involves storing malicious scripts as posts within vulnerable social networking web application and forums, affecting every user accessing the compromised post with embedded malicious code. This problem intensifies when malicious scripts automatically propagate across all the user's social network connections registered under his account [3]. Basically, XSS attacks are mainly caused by the lack of appropriate control mechanisms of user inputs in input forms. This is due to: (1) unfamiliarity of web developers with security issues and practices, (2) business requirements for rapid changes of web services to meet new customers' needs and (3) the simple nature of the attack itself where user inputs should never be trusted and need to be considered as potential attack vectors. XSS attacks can be launched from client and/or server-sides which makes them difficult to be tracked.

Recent statistics underscore the ongoing severity of XSS attacks, posing substantial threats to organizations and individual users. Notably, the XSS vulnerability has consistently appeared on the OWASP's list of top 10 vulnerabilities since 2003, with the most recent report in 2021 reaffirming its significance [4]. Additionally, the Acountix web security tool's annual report for 2021 identifies XSS as the fourth-highest severity vulnerability, detected in 25% of scanned web targets, further highlighting its prevalence [5]. These findings emphasize the critical need for robust measures to address and mitigate XSS vulnerabilities in web applications. It is worthnoting that XSS attacks are not confined to specific browsers, web applications, operating systems, or platforms. Instead, they are generic attacks that can target a wide range of web applications deployed in various web technologies and architectures, including IoT [6], Android and iOS operating systems [7], as well as widely distributed architectures such as cloud-based systems [8–12]. These attacks pose a significant threat across diverse environments, emphasizing the need for comprehensive security measures to safeguard various web infrastructures against XSS vulnerabilities.

Despite the growing awareness of XSS attacks and the efforts to mitigate this threat, the rapid evolution of attack techniques, scripting languages, and the dynamic nature of web applications continue to present significant challenges for web developers and cybersecurity professionals. Traditional defense mechanisms such as filtering input validation based on predefined black or white list of words or tokens may not be sufficient enough to thwart the sophisticated and constantly evolving XSS attack vectors.

Although several reviews on XSS attacks have been published, the most recent high-quality surveys date back to 2020 [13–15]. These earlier surveys were limited in scope, focusing solely on either XSS attacks or vulnerabilities and only considering papers published until 2019. Furthermore, their specific emphasis on tasks such as detection or protection highlights the urgency for an up-to-date, comprehensive, and systematic review that incorporates both previous and latest advancements in the field. Such a review aims to offer a holistic view of the progress in techniques for handling both XSS attacks and vulnerabilities over the years while also providing a comprehensive taxonomy of XSS attacks and their handling approaches. By identifying the gaps left by the earlier studies, this new review seeks to bridge crucial knowledge gaps and contribute substantially to the field of XSS research. To address this need, we conduct a systematic review to collate and analyze the latest research and advancements in XSS attack defenses. By synthesizing the collective knowledge from a wide range of scholarly sources, our review aims to provide valuable insights and guidance for researchers, practitioners, and organizations in devising robust measures to counter the XSS threat effectively.

The aims of the present study can be summarized as:

1. Summarize existing works and study how the interest to XSS attacks evolve over the years.
2. Draw out a comprehensive taxonomy of XSS attacks and their handling techniques.
3. Detect any bias toward specific types of XSS attacks or solution proposals.
4. Identify potential gaps, open challenges and issues in handling XSS attacks.

To achieve the aforementioned aims, we conduct a systematic search for relevant papers published since 1999 following the Kitchenham method for developing systematic mapping [16]. We also present and thoroughly survey existing studies. We describe each category of solutions showing their strengths and weaknesses and we compare similar approaches when possible. This helps researchers and practitioners from both academia and industry in keeping track of existing efforts and provides a comprehensive guide to their future research directions.

The sequel of this paper is structured as follows: Section 2 provides a comparison between the present review and previously published surveys on the topic. Section 3 outlines the research methodology adopted for this review, while Section 4 presents the outcomes of the search and selection processes. Section 5 elaborates on the results of the mapping study, where subsequent Sections 5.1 to 5.5 each addresses one of the research questions leading this comprehensive study. Section 6 discusses the derived conclusions from the review, and Section 7 presents the main findings, comparing them with previously conducted surveys to illustrate the evolution of the state-of-the-art over the years. Section 8 addresses threats to the validity and Section 9 presents the concluding remarks.

2. Related reviews

Several reviews are found in the literature summarizing existing studies on XSS attacks. In this section, existing reviews are presented chronologically and discussed showing their main focus and findings. Those reviews are systematically identified by the same search process used in this study. Table 1 compares those reviews regarding four quality attributes:

1. **Type of the review (type) - regular survey/systematic:** While regular surveys offer insights into individual perspectives about specific topics, they often have limitations in terms of generalizability and potential bias. The preference for systematic reviews over regular surveys has significantly increased in recent years due to several compelling reasons.
2. **Providing taxonomies (taxonomy):** Determining whether a review proposes a classification of approaches, techniques, or tools examined in the included papers is crucial in assessing the review's value and contribution to the field. A well-constructed classification provides a structured framework for organizing and understanding the diverse range of approaches and solutions presented in the literature. It enables readers to grasp the broader patterns and trends in research and facilitates the identification of common topics and gaps in the current body of knowledge.
3. **Scope of the review (scope):** Checking whether the review is specifically focused on a particular aspect of handling XSS attacks or if it encompasses all aspects related to the topic. It determines the extent and depth of coverage of the review, indicating whether it concentrates on a narrow subset of the subject or provides a comprehensive examination of all relevant aspects. Two main points are considered in this case: (1) addressing attacks and/or vulnerabilities, (2) the aspect addressed by the review (i.e., detection, prevention, defense or protection, mitigation).

Table 1
Comparison of existing reviews: Y: provided, N: not provided. R: Regular survey, S: Systematic. A: Attacks, V: Vulnerabilities.

Study	Year	Year	Taxonomy	Scope		Coverage
				A/V	Aspect	
Malviya et al. [17]	2013	R	Y	AV	All	2003–2012
Hydara et al. [18]	2015	S	Y	AV	All	2000–2012
Nithya et al. [19]	2015	R	N	AV	Detection & Defense	2001–2009
Deepa and Thilagam [20]	2016	S	Y	V	Detection & Prevention	2005–2015
Gupta et al. [21]	2017	R	Y	A	Detection	2006–2015
Chaudhary et al. [22]	2018	R	N	A	Defense	2006–2015
Sarmah et al. [23]	2018	R	N	AV	Detection	2002–2017
Gupta and Gupta [13]	2019	R	N	A	Defense	2004–2018
Liu et al. [14]	2019	R	N	V	Detection	2013–2019
Rodríguez et al. [15]	2020	R	Y	A	Defense	2013–2019
This work	2023	S	Y	AV	All	1999–2022

4. **Coverage:** The range of years in which primary papers were published plays a critical role in determining the recency of a review and its findings. It reflects the timeframe during which relevant literature was considered, allowing readers to gauge the currency and relevance of the research findings presented in the review.

Malviya el al. [17] classified existing XSS attack defensive solutions into avoidance, prevention, detection and removal. According to the authors, avoidance can be achieved through enforcing input sanitization where prevention can be attained by adopting guidelines from trusted and well-known community projects such as OWASP. Three detection and removal approaches are discussed in the survey: static, dynamic and hybrid approaches. The authors ended up by advocating the use of advanced program analysis, specifically artificial intelligence based solutions that may provide a trade-off between performance, amount of manual work and suitability to discover new attacks. Although Malviya et al. [17] review comprehensively covers various aspects of handling XSS attacks, it is essential to note that their review is now outdated and does not adhere to the systematic review methodology. Therefore, there is a need for an up-to-date and systematic review to incorporate the latest advancements in the field and provide a comprehensive assessment of the current state-of-the-art in tackling XSS attacks.

Hydara et al. [18] conducted a systematic literature review on XSS. The review included studies published before 2013 focusing on XSS attacks. They identified proposed techniques to handle the XSS issue, their main focus was the identification of most addressed types of XSS attacks and defense techniques. The review revealed much focus on reflected XSS and XSS vulnerabilities detection but less focus on XSS vulnerabilities elimination. Our review is comparable to the review conducted by Hydara et al. [18]; however, it differs significantly in its approach. While Hydara et al. [18] review heavily focuses on bibliometric analysis and limits its coverage to papers published before 2013, the present review takes a much broader approach. It encompasses published papers both before and after 2013, providing a comprehensive survey of existing approaches in addressing XSS attacks. Our emphasis lies on facilitating a thorough and insightful discussion of relevant papers, ensuring a comprehensive understanding of the state-of-the-art in the field.

Nithya et al. [19] presented a comprehensive survey of existing approaches on the detection and prevention of XSS attacks. A total of 36 techniques are presented with their limitations and deployment locations. The review conducted by Nithya et al. [19] lacks a comprehensive taxonomy of existing approaches. Instead, it simply maps existing studies to traditional classifications for vulnerability detection (static, dynamic, hybrid) and categorizes XSS attacks based on their locations (client, server, hybrid). Furthermore, their review solely focuses on XSS vulnerability detection and attack defense techniques, omitting other crucial aspects. Additionally, the review's inclusion of recent papers is limited, with the most recent publication dating back to 2009, rendering their findings outdated in comparison to the latest advancements in the field.

Deepa and Thilagam [20] presented a broader systematic mapping of defense mechanisms against injection and logic vulnerabilities including SQL injection and XSS flaws. In the mapping, studies are categorized based on the phase of software development life cycle where the defense mechanism is integrated. The authors revealed that there is no single solution to mitigate all the flaws. They advocated much focus on fixing vulnerabilities caused by the source code of web applications. The systematic mapping conducted by Deepa and Thilagam [20] primarily focused on injection vulnerabilities in a general context, including but not limited to XSS, and specifically covered detection and prevention techniques. In contrast, our study has a specific and exclusive focus on XSS, encompassing all aspects related to handling XSS attacks and vulnerabilities comprehensively.

Gupta et al. [21] proposed a taxonomy for XSS attacks putting much emphasis on XSS worms. State-of-the-art-techniques are qualitatively compared regarding their deployment locations, strengths and weaknesses and handled attacks. The study revealed a lack of appropriate DOM-based attack detection. In the other hand, Chaudhary et al. [22] provided a similar review putting much emphases to the defense mechanisms against XSS attacks. The authors of this last review identified five research gaps deduced from their qualitative comparison: (1) less attention toward DOM and Mutation XSS attacks, (2) inappropriate discrimination of benign from malicious scripts, (3) improper handling of partial script injection (slight modifications to benign scripts), (4) inappropriate context determination and (5) incomplete sanitization support for new HTML5 features. Although both reviews offer a comprehensive discussion of studies published before 2015, it is important to note that their focus is solely on the detection of XSS attacks. Furthermore, Gupta et al. [21] focus on specific type of XSS attacks (XSS worms) and Chaudhary et al. [22] fails to provide a taxonomy of the discussed techniques.

Sarmah et al. [23] discussed existing approaches and tools for the detection of XSS attacks and vulnerabilities. Tools are compared regrading their platforms, deployment locations and availability of graphic user interfaces. The authors concluded the study by proposing ten recommendations including putting much emphasis for the detection of unknown variants of XSS; consider HTML5 attacks and handling multi-step attacks caused by XSS. Although the conclusions are interesting, the review put significant emphasis on comparing existing tools from user perspectives, rather than concentrating on theoretical models and frameworks. The review lacks a comprehensive taxonomy of the discussed tools, which could provide a more systematic and structured understanding of the landscape.

Gupta and Gupta [13] adopted a set of qualitative criteria for comparing existing endeavors and ended up with a list of guidelines for designing robust solutions for XSS attacks. The guidelines include developing techniques for nested context determination and sanitization for new HTML5 features. The nature of the review leans more toward a comparative analysis rather than a comprehensive survey. It lacks specific details regarding the presented studies and focuses solely on XSS attack defense techniques without providing a taxonomy of presented works.

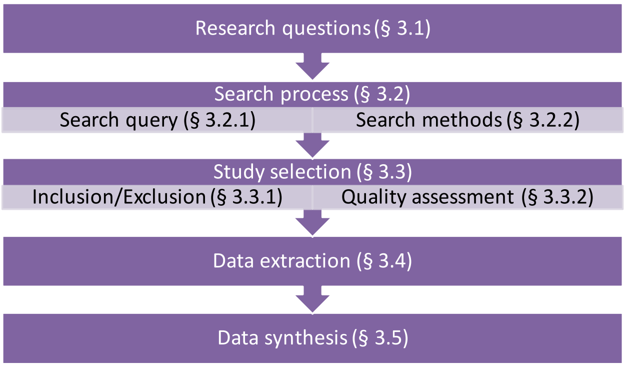

Fig. 1. Overall steps for conducting the systematic mapping.

Liu et al. [14] also adopted the traditional classification of existing works on the detection of XSS vulnerabilities (static, dynamic and hybrid). The authors discussed the pros and cons of selected studies together with their abilities to detect different types of XSS attacks. The authors found that no method is perfect and recommended the combination of different techniques at the same time. The scope of this review is restricted to XSS vulnerability detection, mirroring the limitations of the three previously discussed reviews, which also lack a taxonomy of the featured works.

Rodríguez et al. [15] summarized tools and methods to mitigate XSS attacks. Selected tools and methods are compared regarding different qualitative attributes including their reliance on artificial intelligence techniques. The study revealed a low trend for the use of machine learning compared to traditional methods. Therefore, they advocated the use of artificial intelligence techniques and investigation of the impact of XSS attacks on Internet of Things and advanced industrial devices. Rodríguez et al. [15] review's scope is confined to XSS attack defense techniques, and its primary focus revolves around a single research question, which explores the inclination of classic technology proposals compared to the adoption of artificial intelligence for mitigating XSS attacks.

3. Research methodology

In this section, we present and detail the review protocol adopted for the elaboration of the present study. We followed the reported guidelines of Kitchenham et al. [16] and Kuhrmann et al. [24] for the preparation and validation of the protocol. In the subsequent subsections, we describe the set of research questions pointing out the focus of the study, the elaborated process for searching, filtering, and selecting relevant papers, and the data extraction and synthesis processes. The complete protocol adopted for conducting the present study is illustrated in Fig. 1, along with the corresponding section number for each step.

3.1. Research questions

The purpose of this paper is to review existing studies with a primary focus on handling the cross-site scripting issue. This leads to identifying current gaps and revealing potential future research directions. Research questions are formulated in light of these aims, and the study is conducted with five main questions in mind:

RQ1. How has interest in addressing XSS attacks evolved since its unveiling in 1999? This research question is adopted to check whether XSS has gained sufficient attention regarding its prevalence.

RQ2. What type of researches have been published addressing the XSS issue? This question aims to identify current research focused on tackling XSS flaws.

RQ3. What types of XSS attacks are addressed by contemporary studies? By answering this question, we aim to provide a classification of existing XSS attacks and identify the most commonly addressed XSS attack types.

RQ4. What techniques have been used to tackle the XSS issue? This question is considered to provide a global overview of already explored techniques, approaches, and technologies handling XSS attacks and vulnerabilities, as well as their scopes and applicability levels.

RQ5. What types of evidence are used to validate solution proposals? This question enlightens the type of validation techniques and sources of experimented data, if any.

3.2. Search process

For retrieving relevant papers for our study, we adopt a hybrid approach incorporating automatic and manual search methods. In this section, we provide an overview of how candidate papers were searched.

3.2.1. Search query

The search string used for the automatic search in digital libraries was developed in accordance with the guidelines established by Petticrew and Roberts [25]. Following these guidelines, we identified population terms that are relevant to our area of interest, which in this case is "XSS". However, intervention terms were omitted as our focus is not on a specific intervention. Instead, our objective in this review is to include any study where the primary focus is XSS. Consequently, the final search string adopted for our review consists of the term "XSS" or any of its variations and associated names extracted from previous reviews discussed in Section 2. By adopting this approach, we aim to ensure comprehensive coverage of relevant literature pertaining to XSS. The final adopted search string is:

"XSS" OR "Cross-site scripting" OR "Cross site scripting"

We observed that many papers address generic attacks but refer to XSS in their abstracts as a comparable attack or vulnerability. To reduce the number and increase the relevance of the retrieved papers, we decided to restrict our search to paper titles. This way, only papers with a main focus on XSS are retrieved.

3.2.2. Search methods

The above search query is used for automatic search. The selection of digital libraries is a crucial step in conducting systematic reviews. In our search for high-quality peer-reviewed papers, we initially conducted an automated search using Scopus. However, it is important to note that index databases such as Scopus and WOS may not encompass all relevant papers [26], and the timeframe for papers to be indexed can vary even for journals and conference proceedings already included in their lists. Factors such as the publication schedule, submission volume, and indexing process efficiency influence the indexing timeline. Consequently, relying solely on index databases may result in overlooking recently published papers that have not yet been incorporated into these databases. It is also important to note that while adopting only index databases is recommended for rapid reviews, it may not suffice for comprehensive systematic reviews [27]. To ensure a comprehensive approach and avoid missing any relevant studies, we adhered to the guidelines provided by Kuhrmann et al. [24]. It is recommended to consider not only index databases but also to explore individual academic libraries. In our searching process, we selected Scopus as an index database due to its wider and more inclusive content coverage compared to WOS, as revealed by the recent Pranckutė bibliometric analysis [28]. Consequently, the following six online academic libraries are explored:

1. Scopus (https://www.scopus.com/)
2. IEEE Xplorer (https://ieeexplore.ieee.org)

Table 2
Inclusion and Exclusion criteria: I_i : inclusion criterion, E_i: exclusion criterion.

ID	Criteria
I1	papers addressing any aspect regarding XSS attacks and/or vulnerabilities
I2	papers written in English
I3	peer-reviewed journal or conference papers
I4	papers with full text available
I5	publications published until 2022
E1	papers not addressing XSS such as general-purpose attacks or vulnerabilities detection tools or methods.
E2	duplicate papers (older versions of already retrieved papers) and short papers with the presence of extended versions. Snowballing is performed for old and new versions to avoid missing relevant papers.
E3	white papers, technical reports, reviews/surveys, patents, feature articles, very short papers (less than 3 pages length), books and book chapters.

3. ACM Digital Library (https://dl.acm.org)
4. SpringerLink (https://link.springer.com)
5. ScienceDirect (https://www.sciencedirect.com/)
6. Wiley Online Library (https://onlinelibrary.wiley.com)

To alleviate the impact of the above taken decision about word search position and to avoid potential missing of relevant studies, we conducted a recursive backward and forward snowballing on approved papers as suggested by Wohllin [29]. Therefore, references and citations in/of approved papers are screened for relevance regardless of the presence of search strings in their titles. The snowballing is repeated for each new approved paper. Google Scholar is used for exploring and citing papers. In order to maintain objectivity and minimize bias in the selection of relevant papers, we have taken measures to ensure the inclusion/exclusion criteria (Table 2) and quality assessment criteria (Table 3) are as objective as possible, facilitating replication of the study. Moreover, we adopted a two-step screening process. Two different authors independently screened the papers, and in instances where conflicts arose, a dedicated discussion session was conducted to resolve any discrepancies.

3.3. Study selection process

The list of found papers was subject to two separate screening stages. Firstly, the metadata, including titles and abstracts, is read and checked for relevance and whether it meets a subset of related inclusion and exclusion criteria. Secondly, the full texts of papers are examined to check if they meet the rest of the inclusion and exclusion criteria and achieve a reasonable quality assessment score. Metadata screening is performed separately by the two first authors; decisions are exchanged, and conflicts are discussed and solved.

3.3.1. Inclusion and exclusion criteria

A set of inclusion and exclusion criteria are considered for the selection of appropriate and relevant papers. In this study, only English published journal and conference papers with available full texts are selected. The search process is conducted in October 2021; to avoid missing interesting papers, we searched papers published since 1999, which is the year where the first XSS issue was reported by Microsoft security engineers, we also considered early publications for complete search. The entire list of inclusion and exclusion criteria adopted for the selection process, in this study, are described in Table 2.

3.3.2. Quality assessment

To ensure the inclusion of high-quality studies, a strict quality assessment check was conducted on all selected papers from the previous stage. A checklist of 5 items was employed to assess the quality, and only papers with a total score greater than or equal to 4 were considered (see Table 3 for details). The design of the quality assessment criteria followed the guidelines of Zhou et al. [30], focusing on four key aspects: relevance, reporting, rigor, and credibility [30]. The first criterion (QA1) considers the relevance aspect. It prioritizes studies focusing solely on XSS, giving higher scores to papers addressing this issue without considering similar problems like SQL injection. QA2 deals with the reporting aspect since it evaluates the clarity of the proposed solution and appropriateness of the adopted methodology reported in each study. Papers with insufficient self-containment or lacking essential details receive a score of 0.5 or 0, based on the provided information. QA3 assesses the rigor of the study by evaluating the appropriateness of the evaluation process for the proposed solution. Papers that use a reasonable test case or dataset size, compare with related works, and employ appropriate evaluation metrics receive higher scores in this criterion. For credibility, QA4 checks the publication venue's popularity, with paper sources ranked based on CORE conference ranking (A or B: +1, C: +0.5, not ranked: 0) and SJR rankings (Q1-2: +1, Q3-4: 0.5, not ranked: 0). Finally, QA5 evaluates the impact and influence of each study on the research field. Papers with a number of citations greater than or equal to the number of years since publication receive a score of 1, while others receive a score of 0, ensuring that newly published studies are not penalized. Google Scholar is used to identify the number of citations associated with each paper. To achieve balance among the four aspects (relevance, reporting, rigor, and credibility), the weights assigned to each criterion and the eligibility score were carefully selected. This approach ensures no single criterion becomes decisive for the approval of papers, promoting a fair and comprehensive evaluation. The quality assessment is performed separately by the first two authors, and any conflicts are resolved through discussion.

3.4. Data extraction process

Data are extracted manually from selected and approved papers by the first two authors and validated by the third author. Disagreements are resolved by discussion to reach a consensus. In this phase, we followed the guidelines of Peterson et al. [31] where a data extraction form is designed as illustrated in Table 4. Each selected paper is thoroughly analyzed by the authors, where all the required data for answering the research questions presented in Section 3.1 is extracted and detailed in the form. Besides the metadata used for answering RQ1, more detailed information required for answering the other research questions is made explicit in the designed form. Those details include the type of study, a detailed description of the proposed solution or analysis process, the applicability level of the proposal, if any, techniques, and lists of data sources used for validation.

3.5. Data synthesis

After data extraction from approved papers, descriptive statistics and frequency analysis are performed to answer research questions RQ1, RQ2, and RQ5. The results are illustrated using tables and visualized using appropriate plots and graphics. In addition, a thematic analysis is performed based on the guidelines suggested by Cruzes et al. [32] to answer RQ3 and RQ4. Accordingly, patterns in the data are first identified and then grouped into distinct themes; those themes are reviewed and refined to produce the final themes. The three authors are involved in the elaboration of this task.

4. Results of the search and selection processes

Two search rounds were carried out. The first round was conducted in October 2021, retrieving all papers published with a focus on XSS, including early publications from 2022. The second search round was performed in July 2023 to include any papers published in 2022 that might have been missed in the first round. Fig. 2 provides an overview of the search and selection processes, illustrating the number of identified, included, and excluded papers at each step. The black

Table 3
Quality assessment criteria.

ID	Criteria	Value
QA1	Does the study exclusively focus on XSS attacks and vulnerabilities?	Yes(1)/No(0.5)
QA2	Does the study present a clear solution or adequate research methodology?	{0, 0.5, 1}
QA3	Does the study used appropriate methods to validate the proposed solution?	{0, 0.5, 1}
QA4	Has the study been published in a well-established source (journal or conference)?	{0, 0.5, 1}
QA5	Has the study been fairly cited by other papers?	{0, 1}

Table 4
Data extraction form.

ID	Data item	Description	RQ
D1	Year	year of the publication	RQ1
D2	Source	venue of the publication	RQ1
D3	Type of publication	conference or journal paper	RQ1
D4	Type of the study	analysis, solution proposal	RQ2
D5	Focus	attacks, vulnerabilities	RQ2-4
D6	Description	a short overview of the proposed solution	RQ2-4
D7	Taxonomy	proposed or adopted classification or taxonomy, if any	RQ3
D8	Types of XSS	type of XSS attack(s) addressed by the study	RQ3
D9	Detailed focus of the study	prevention, detection, protection, etc.	RQ4
D10	Applicability	applicability level of the solution: client, server, proxy, etc.	RQ4
D11	Evaluation	validation process adopted by the study	RQ5
D12	Sources	list of datasets or sources used for validation if any.	RQ5

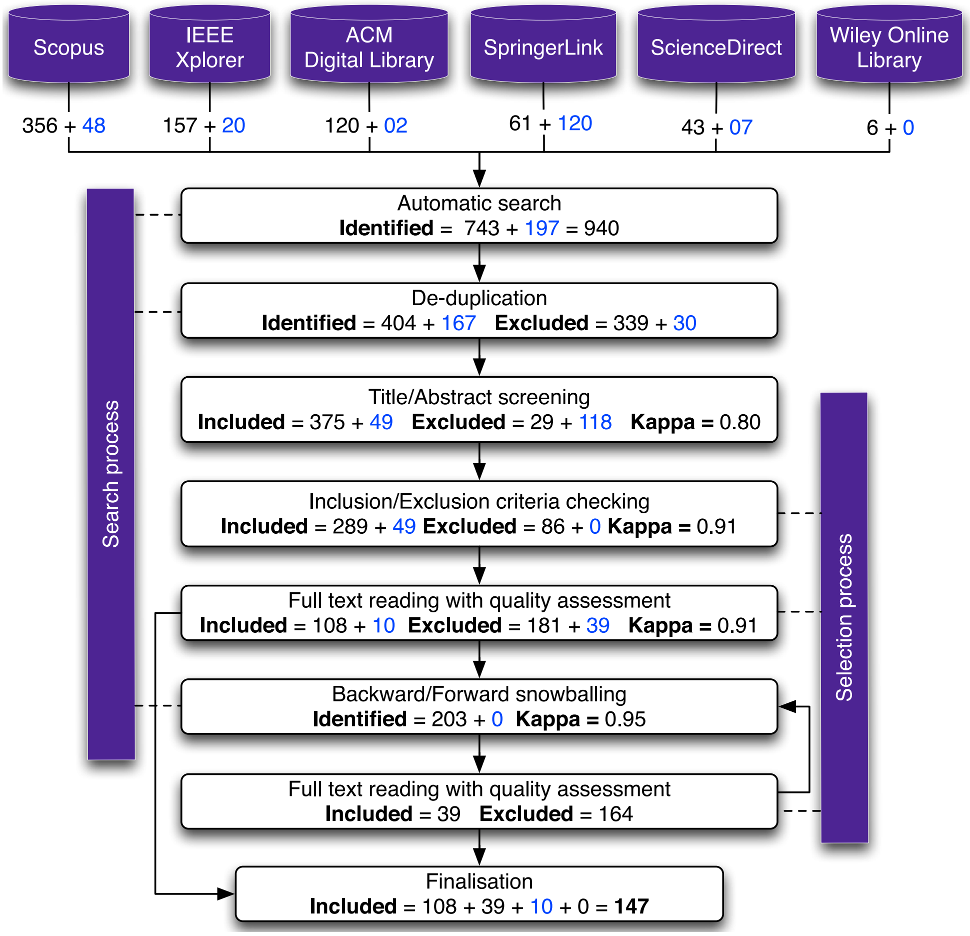
Scopus
IEEE Xplorer
ACM Digital Library
SpringerLink
ScienceDirect
Wiley Online Library
356 + 48
157 + 20
120 + 02
61 + 120
43 + 07
6 + 0
Search process
Automatic search
Identified = 743 + 197 = 940
De-duplication
Identified = 404 + 167 Excluded = 339 + 30
Title/Abstract screening
Included = 375 + 49 Excluded = 29 + 118 Kappa = 0.80
Inclusion/Exclusion criteria checking
Included = 289 + 49 Excluded = 86 + 0 Kappa = 0.91
Full text reading with quality assessment
Included = 108 + 10 Excluded = 181 + 39 Kappa = 0.91
Backward/Forward snowballing
Identified = 203 + 0 Kappa = 0.95
Full text reading with quality assessment
Included = 39 Excluded = 164
Finalisation
Included = 108 + 39 + 10 + 0 = 147
Selection process

Fig. 2. Results of the search and selection processes.

numbers represent results from the first search round, while the blue numbers represent results from the second search round.

The search process began by exploring six different academic search engines using the search query described in Section 3.2. In the first search round, this step yielded 743 papers, while in the second search round, it yielded 197 papers. After de-duplication, 404 distinct papers remained from the first round, and 167 remained from the second round. The identified papers from the automatic search were then subjected to an eligibility check. During the title and abstract screening, 29 papers from the first round and 167 papers from the second round were found to be irrelevant to XSS. Additionally, 86 more papers from the first search round did not meet our inclusion and exclusion criteria.

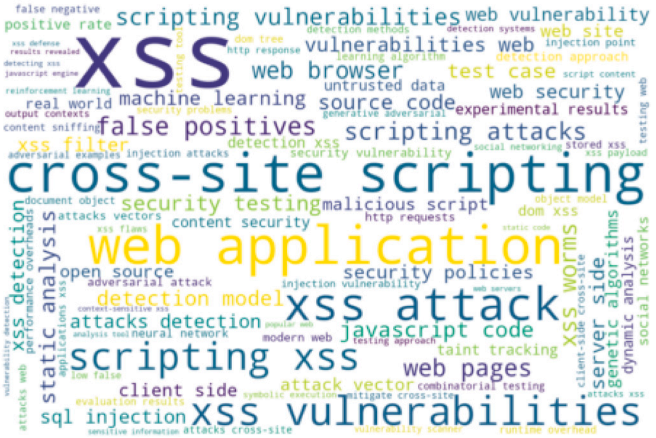

Fig. 3. Word cloud constructed using titles and abstracts of included studies.

After this screening process, only 108 papers from the first round and 10 papers from the second round remained, meeting the threshold value adopted for the quality assessment check (i.e., having quality scores greater than or equal to 4). These papers formed the initial set of approved papers. The recursive backward and forward snowballing process over the 108 approved papers from the first round identified 203 new papers, but only 39 of them passed the eligibility check. Additionally, a paper previously identified in the first search round was replaced by an extended version published by the same author, in accordance with the E2 criterion (see Table 2). This resulted in a total of 157 papers. The snowballing phase performed on the second round of searches did not reveal any new papers. In the selection process, the de-duplication step was made automatically getting benefits of the BibDesk[1] tool's available feature. Since the selection is separately performed by the two first authors, Cohen's Kappa coefficient was calculated to measure inter-reviewer agreement at each selection stage as suggested by Kitchenham et al. [16]. The Kappa scores were 0.80, 0.91, 0.91, and 0.95 at each stage as depicted in Fig. 2.

As suggested by Kuhrmann et al. [24], a Word Cloud of frequent terms used in titles and abstracts of included papers is shown in Fig. 3. Terms are firstly extracted using 1-2grams and then filtered to keep only significant ones. For preserving the visibility of the figure, only top 100 terms are kept. Fig. 3 illustrates that *XSS*, *cross-site scripting* and *web application* are the most frequent terms, which is consistent with the adopted terms in the search query. Moreover, the figure illustrates a similar interest on XSS attacks and vulnerabilities as well as static and dynamic analysis and client and server side (have quite similar size in the cloud). Moreover, the word *detection* appears frequently in conjunction with terms such as *model*, *attacks* and *XSS*; this indicates that much focus is given to the detection of the problem compared with other techniques such as protection. However, the validation of these preliminary observations requires more exhaustive analysis which is the focus of the proposed research questions steering the present review.

5. Results of the mapping study

In this section, we present the main results of the mapping study, addressing the research questions that motivated the present review. Additionally, we provide potential taxonomies and offer a comprehensive discussion of the studies retrieved through the search process.

5.1. How has the interest in addressing XSS attacks evolved since its unveiling in 1999?

The adopted search and selection processes resulted in the retrieval of 157 distinct, relevant and high quality papers. Fig. 4 shows the distribution of included papers according to their publication years and publishers. Remarkably, although the early detection of the XSS issue in 1999, the attack started earning academic research attentions since 2016 and higher attentions were given in the last years (since 2019).

Table 13 (see the appendix) shows the entire list of included studies with their publication years and their correspondent detailed quality assessment scores. The studies in the table are ordered descendingly following their publication years.

To provide a more comprehensive analysis of XSS attack research interests, we conducted a bibliometric analysis using the bibliometrix tool [33]. This analysis enables the examination of the type of publications, the most relevant sources of publications, the countries of corresponding authors, and the author affiliations. The study revealed that only 39 papers have a total quality assessment score where 56 papers have the exact threshold value for eligibility. Specifically, 33 papers have been published in low ranked journals or conferences and 8 of them are penalized since they address other issues in addition to XSS, the others do not completely satisfy QA2 and QA3 (i.e.; do not present a clear solution and have lacks in their adopted validation methods). Most included studies (101/157) are conference papers and only 56 from the entire set are journal publications (see Fig. 5). This is due to the publication speed of conference papers compared with journals [34]. Conferences are also a good venue for facilitating face-to-face interactions amongst scholars working in the same field. This is a more practical way to receive timely feedback and enhance outcomes in continuously growing fields of computer science like web security [34]. Another factor to take into account is the existence of a number of cybersecurity leading conferences, which encourages researchers to submit their work to these events and spread their findings to the security community as quickly as they can. Table 5 shows the top leading conferences where included studies are published. Moreover, as stated earlier, research on XSS has only lately (since 2016) gained prominence, thus it can take some time until journal publications on XSS appear. The highest number of conference papers explains why the top three publishers of included studies are IEEE (41 papers), ACM (30 papers) and Springer (24 papers), since they are pioneers in the publication of conference proceedings. Fig. 5 indicates an increasing number of published journal papers, in recent years, compared with conference papers. Elsevier, with 23 papers, is the largest publisher of journal papers on XSS in the last two years where 8 of them come from the Computer & Security journal; one of the most relevant with a fast review process (average of 7.5 weeks from submission to first decision in 2021)[2].

Fig. 6 displays the top 20 corresponding author countries, while Fig. 7 presents the top 20 author affiliations. The bibliometric analysis revealed that among the countries contributing to XSS attack research, China, USA, and India stood out. The higher number of publications from these countries could be attributed to various factors, such as the large and active research community, government initiatives and cybersecurity landscape. The collaborative nature of cybersecurity research, especially concerning XSS attacks, emphasizes the importance of international cooperation among researchers and institutions. Fig. 6 clearly illustrates this aspect. Notably, a significant proportion of papers published by Chinese researchers are the outcomes of collaborations with foreign institutions. Moreover, it is noteworthy that all the papers originating from Greece and Argentina are the results of collaborations as well. This trend highlights the global effort to collectively address and tackle security threats, emphasizing the

[1] BibDesk Bibliography manager: https://bibdesk.sourceforge.io

[2] https://journalinsights.elsevier.com/journals/

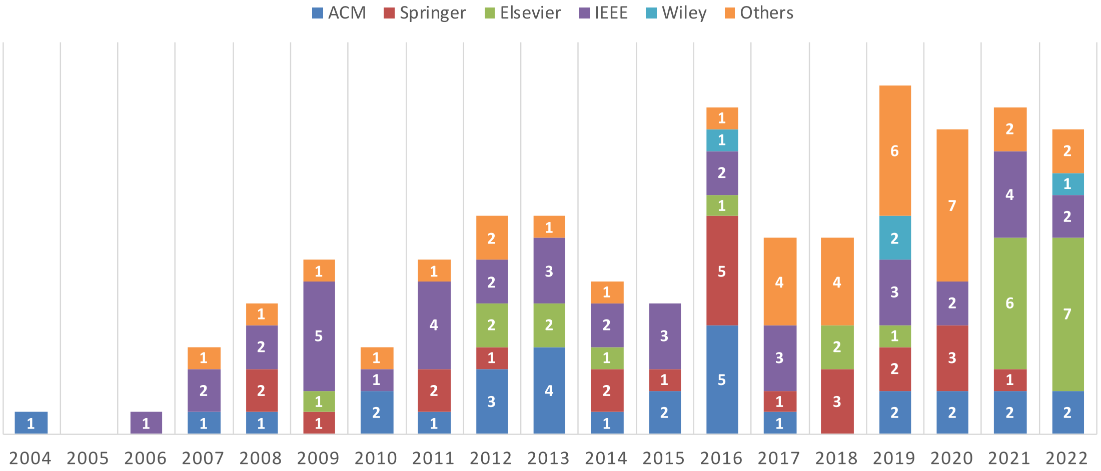

Fig. 4. Distribution of included studies over years and publishers.

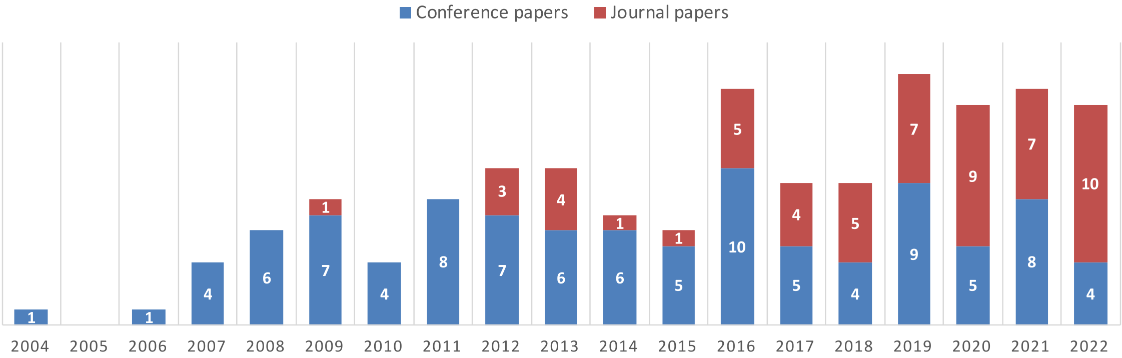

Fig. 5. Distribution of conference and journal papers over years.

Table 5
Top conference proceedings publishing research on XSS.

Acronym	Conference name	# included studies
CCS	ACM Conference on Computer and Communications Security	8
ESORICS	European Symposium on Research in Computer Security	6
WWW	International conference on World Wide Web	6
SP	IEEE Symposium on Security and Privacy	5
NDSS	Network and Distributed System Security Symposium	5
ASIACCS	ACM Asia Conference on Computer and Communications Security	4
ICSE	ACM/IEEE 30th International Conference on Software Engineering	4
USENIX	USENIX Security Symposium	3

significance of international partnerships in advancing cybersecurity research and fostering a safer digital environment. This indicates that XSS attack research has garnered significant attention and contributions from researchers in these countries. When examining the author affiliations, three institutions emerged as major contributors to XSS attack research. Sichuan University, University of California, and National Institute of Technology Kurukshetra.

5.2. What type of researches are published addressing the XSS issue? (RQ2)

Identified studies can be classified regarding their intents and research types into two main categories:

1. *Analysis/experimentation:* studies of this category focus on examining the impact of the XSS issue and comparing state-of-the-art approaches highlighting their strengths and weaknesses. They do not actually propose solutions to the XSS issue, but they may end up with a set of guidelines or hints on how to tackle the issue. Their main focus is one of three kinds: (1) examine the prevalence of the attack or one of its variants in the wild (e.g., Heiderich et al. [35], Melicher et al. [36], and Steffens et al. [37]), (2) identify the attack underlying features in the aim to understand its particularities and provide proper countermeasures (e.g., Chaliasos et al. [38] and Zhang et al. [39]), and (3) compare existing solutions highlighting their achievements and unfulfillments regarding a set of predefined qualitative and/or quantitative criteria (e.g., Weinberger et al. [40], Faghani and Nguyen [41], and Lekies et al. [42]).
2. *Solution proposal:* compared with the former category, studies in this category provide clear and detailed solutions armed with reasonable proof of their validity. Those studies can also be classified according to their contributions to handle XSS attacks into:

   (a) *Attack prevention based solutions:* focus on the elimination of the source causes of the attack. This is mainly achieved through handling XSS vulnerabilities and can be divided into:

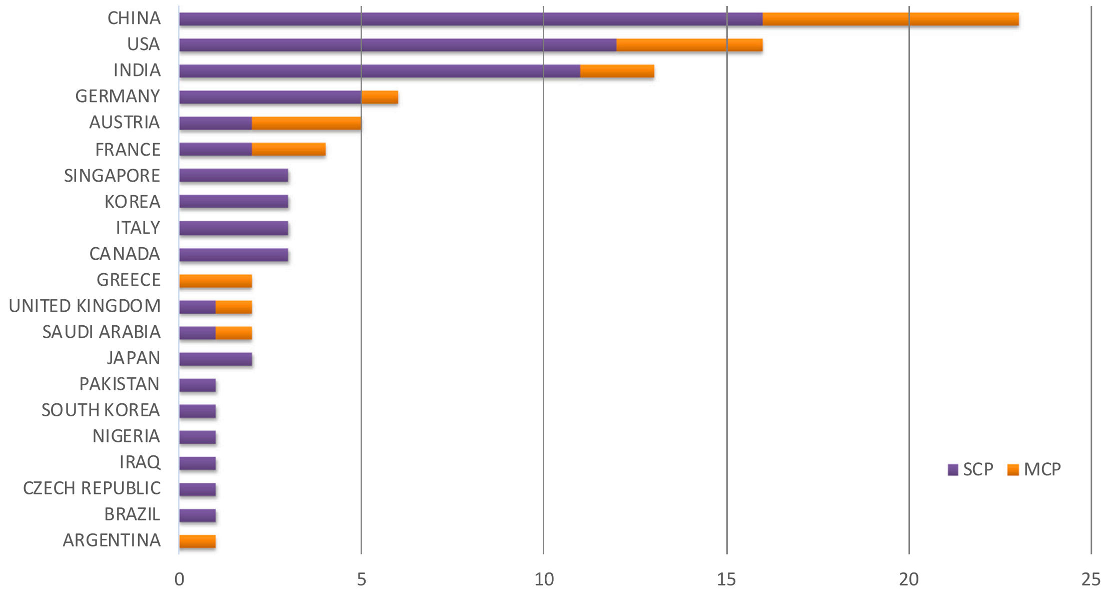

Fig. 6. Most relevant corresponding author's countries: SCP: Single Country Publications, MCP: Multiple Countries Publications.

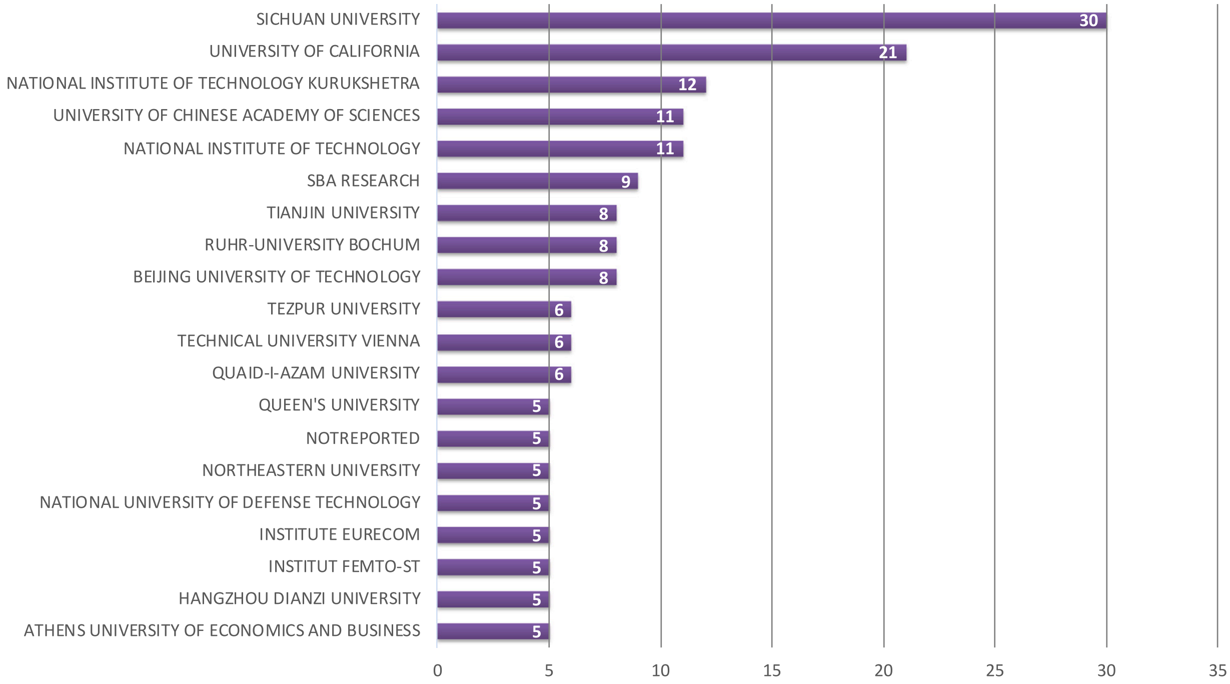

Fig. 7. Most relevant authors affiliations.

i. *Vulnerability mitigation based solutions:* provide and examine a set of guidelines or practices that should be preformed at early stages of the web applications development process (e.g., Wang et al. [43]).
ii. *Vulnerability detection proposals:* enable checking the presence of insecure artifacts, in already developed web applications, prior deployment (e.g., Leithner et al. [44]).
iii. *Vulnerability protection proposals:* propose runtime actions to be taken in order to protect sensitive artifacts of web applications from being affected by malicious input data (e.g., Gupta et al. [45]).
iv. *Vulnerability detection and repair based solutions:* enable dynamic detection and repair of applications vulnerabilities when they occur (e.g., Marashdih and Zaaba [46]).

(b) *Attack detection based proposals:* focus on how to distinguish attacks from normal behaviors on already deployed and active web applications. Detected attacks are generally reported to web application administrators or victims to take appropriate actions (e.g., Tariq et al. [47]).
(c) *Attack defense based solutions:* define a set of countermeasures taken automatically to thwart XSS attacks when they occur. They either provide means to defend against already known attacks or attack vectors or obstruct anomaly behaviors (e.g., Xu et al. [48]).
(d) *Attack detection and defense proposals:* propose a combination of techniques from detection and defense to protect

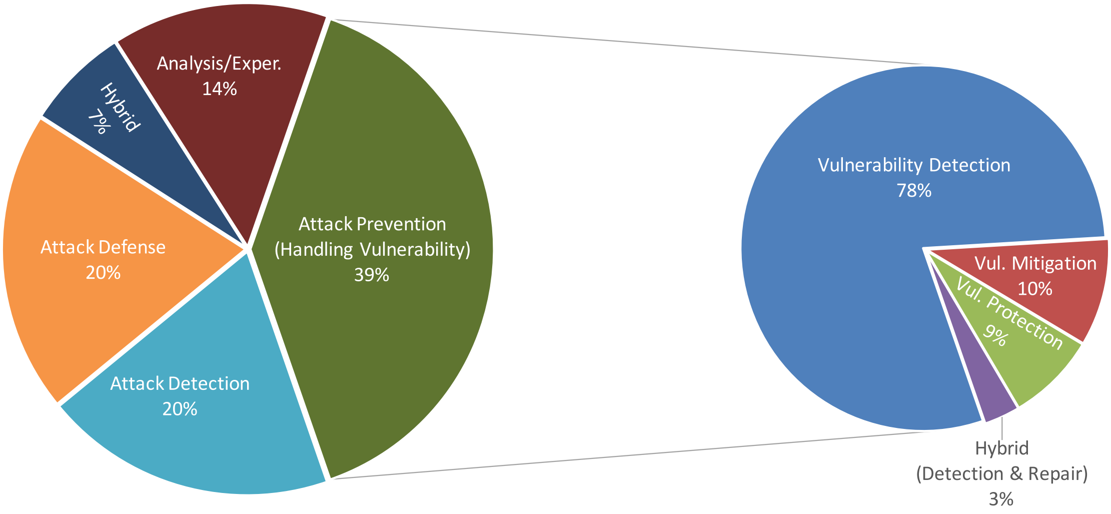

Fig. 8. Distribution of included studies per research focus.

web applications against XSS attacks at runtime. Those solutions, generally, enable the detection of new attacks and act differently according to each observed behavior (e.g., Chaudhary et al. [49]).

Fig. 8 illustrates the distribution of studies regarding the set of distinguished categories.

As shown in Fig. 8, the search process revealed much interest to XSS attack prevention compared with defense and detection. This is a positive fact since prevention is always better than cure. However, much attention is given to vulnerability detection instead of protection and repair. The less interest to XSS vulnerability mitigation techniques is understandable since those tasks require to seduce developers to respect security standards and follow strict security coding practices. Actually, this cannot easily be achieved due to time and cost constraints. The interest to vulnerability detection is also important since it is a first step toward attaining effective solutions. However, proposed solutions should be practical enough to be adopted by less experimented developers and respect time to market. This will be checked when answering RQ3. In the other side, XSS attack defense and detection are earning much attention compared with analysis and hybrid solution proposals. Runtime protection is important but more analysis studies are required to: (1) understand the peculiarity of new XSS attacks, (2) keep track of the development of hackers' tactics and (3) check the suitability of existing proposals to different types of XSS attacks. Table 6 summarizes the focus of each analysis study and gives insights on their major findings.

Table 6 shows that existing analysis and experimentation studies cover several aspects that help in tackling XSS attacks. They vary from studying the prevalence and peculiarities of XSS attack variants to comparing mitigation techniques and defense tools. However, more fundamental analysis studies are still needed to explore the missing aspects. Fundamental analysis studies should provide a deep understanding of XSS artifacts and explain why such attacks are still effective and recurrent. Those studies may include tracking the development of XSS payloads with regard to emerging advanced web technologies and new programming styles. New technologies, such as Web services and APIs, enable complex interactions and data exchanges between clients and servers in the network. This may cause existing defense techniques to fail in the detection of XSS attacks targeting such environments and their applications. New programming styles, on the other hand, such as AJAX, CSS3, and HTML5, introduce new and complicated injection flaws to web applications, making advanced web applications more vulnerable to even primitive XSS attacks.

5.3. What type of XSS attacks are addressed by contemporary studies? (RQ3)

Several XSS attack variants have been identified, explored and studied in the literature. A widely adopted classification is the one reported by OWASP [63] that considers the location where untrusted data are supplied and processed: *client-side* and *server-side* XSS. This classification is adopted by most included studies [8,37,47,53,64,65]. In this review, we propose a more comprehensive classification based on the source of the vulnerability causing the attacks. Accordingly, three categories can be distinguished: *Application-based*, *Third-party-based* and *Collaboration-based*. Fig. 9 shows the proposed classification of XSS attacks together with the number of studies addressing each attack. In the sequel we give a detailed description of each category with the list of their included attacks.

5.3.1. Application-based XSS attacks

Attacks of this category are mainly caused by web applications themselves. Inexperienced developers with security issues often omit the integration of proper sanitization to every user controlled sources. Five of existing attacks fall in this category including the three basic OWASP attacks reported in [63]:

1. *Reflected XSS (RXSS)*: RXSS attacks occur when malicious data supplied as part of HTTP requests become part of server responses without being properly sanitized. Consequently, malicious data embedded in HTTP responses reach back the client-side and get executed when rendered by browsers. Typical targets of RXSS are web applications with search capabilities, where embedded malicious data become part of search results or error messages. These attacks are also referred to in [63] as non-persistent or Type I. They are said to be non-persistent because malicious data are not permanently stored in the server.
2. *Stored XSS (SXSS)*: SXSS attacks appear when malicious data are supplied in input forms to be stored in server application databases or files without being sanitized. This way, users requesting data from those applications are supplied with server responses containing malicious data that become harmful when being rendered by browsers. Typical targets of SXSS attacks are online social web applications and forums where malicious data can be posted, stored in databases and hence infect every user accessing them. SXSS attacks are also referred to in [63] as persistent or Type II. They are persistent because malicious data stored in the server remain harmful and effective until being detected, removed or filtered.

Table 6
Summary of analysis studies.

Study	Focus	Summary of findings
Weinberger et al. [40]	evaluation of sanitization techniques adopted by 12 popular frameworks and web applications to defend against XSS attacks.	• limited support for context-sensitive auto-sanitization. • no support for DOM-based XSS attacks that are dynamically generated at the client-side. • no appropriate sanitization to all available contexts where users should manually develop their own sanitizers.
Scholte et al. [50]	study the relationships between programming languages and vulnerabilities of web applications to XSS and SQL injections.	• PHP web applications are the most popular and the most vulnerable to XSS and SQL injections. • SQL and XSS injections can often be prevented by enforcing common data types on input parameter types.
Faghani and Nguyen [41]	experimentation of analytical models that characterize the propagation of XSS worms in Online Social Networks (OSN).	• visiting friends more than strangers slows down the propagation. • the presence of large number of cliques in the network decreases the propagation speed. • monitoring highly clustered structures of an OSN enables the containment of XSS worms within a community. • early detection of XSS worms can be achieved through monitoring a small number of users selected based on *Degree* and *PageRank* metrics.
Avancini and Ceccato [51]	comparison of genetic algorithms (GA) and concrete symbolic execution (CSE) as two test-case generation for the detection of XSS vulnerabilities.	• GA based test-case generation takes less time to generate a test case and parameter tuning has no significant impact. • CSE has higher coverage rate compared with GA due to the embedded sanity check. • an alternate composition of GA and CSE enables a trade-off between coverage rate and generation time.
Heiderich et al. [35]	study the prevalence of mutation XSS attacks (MXSS) and the suitability of existing techniques to defend against them.	• web-mailers are preferable targets to MXSS attacks. • most existing defense techniques are far away from the effective detection against MXSS attacks.
Bozic et al. [52]	comparing the efficiency of IPOG and IPOG-F algorithms of combinatorial testing regarding the detection of XSS vulnerabilities.	• IPOG-F performs better than IPOG since it generates more comprehensive and sophisticated attack vectors. • adding constraints to the input parameter model significantly improves the quality of generated attack vectors by the two algorithms.
Stock et al. [53]	identify the root causes of client-side XSS attacks and classify them according to the complexity of their tainted-flows.	• 64% of existing flaws are simple and only caused by the lack of security awareness of developers. • 21% quite complex flaws are caused by third-party libraries. • 15% complex flaws are caused by a combination of both causes.
Zhang et al. [39]	introduce XSS attacks caused by the use of insecure API implementations (XAS).	• the examination of 143 third-party applications for social networks showed that all are vulnerable to XAS attacks. • a set of countermeasures may prevent XAS attacks. These include the need to set proper content-type header in API responses and sanitize user-input data incorporated in API responses.
Weichselbaum et al. [54]	study the efficiency of Content-Security Policies (CSPs) in protecting web applications against XSS attacks.	• 94.72% of real-world adopted CSPs can be bypassed. • relying on domain whitelists in CSPs is not sufficient to protect against trivial XSS attacks.
Lin and Barcelo [55]	study the decidability of a logic conceived for analyzing Mutation XSS (MXSS).	• regular expressions alone are not sufficient for the detection of security vulnerabilities. • a sound logic is required for expressing constraints useful for the analysis of MXSS attacks and vulnerabilities.
Bazzoli et al. [56]	derive a set of recommendations for tackling the limitations of blackbox vulnerability scanners in the detection of reflected XSS vulnerabilities.	• use a diverse set of distinct payloads instead of fuzzing or mutating a small set of arbitrary selected samples. • adopt a context-sensitive based selection of candidate payloads. • select short payloads with a small character set. • regularly update the set of payloads for the detection of new vulnerabilities.
Lekies et al. [42]	examine the ability of existing defense techniques to defend against XSS attacks caused by leveraging script gadgets (CR-XSS).	• the examination of 10 XSS mitigation tools (HTML sanitizers, filters, web application firewalls and Content-Security policies) showed that they are unable to handle CR-XSS attacks.
Melicher et al. [36]	check the prevalence and identify the causes of DOM XSS vulnerabilities.	• 83% of vulnerabilities come from advertising and analytics domains. • DOM XSS vulnerabilities can be eliminated by blocking Ads. • DOM XSS vulnerabilities are also caused by the use of incorrectly implemented templates from templating frameworks.
Steffens et al. [37]	check the prevalence of persistent client-side XSS vulnerabilities.	• more than 8% of the examined 5,000 highest-ranked sites are found vulnerable to persistent client-side XSS attacks. • 4 distinct scenarios are identified showing how client-side storage is used in an insecure manner.
Wijayarathna and Arachchilage [57]	check the usability of OWASP ESAPI sanitizers to prevent XSS attacks from developers' perspectives.	• programmers are unable to identify all the locations where sanitizers should be instrumented. They also accidentally encode inputs instead of outputs. The problem is due to the absence of detailed documentations with enough comprehensive examples. • programmers use wrong encoding methods to encode data due to the absence of validation routines.

(*continued on next page*)

Table 6 (*continued*).

Study	Focus	Summary of findings
Chaliasos et al. [38]	experiment a variant of XSS attacks that are caused by slight modifications on the AST of web pages (JSM-XSS).	• JSM-XSS can easily bypass whitelist scripts based defense mechanisms.
Schuckert et al. [58]	extract code patterns that harden static analyzers' detection of XSS vulnerabilities.	• 19 source code patterns are extracted from open source projects and CVE reports. These include the use of super global variables such as `$_SERVER` and `foreach` on super global variables. • there may still be undetectable code patterns.
Buyukkayhan et al. [59]	capture the development of reflected XSS exploits over 10 years of time.	• most reflected server XSS exploits are clear and obfuscation is rarely used. • the majority of exploits could be blocked by existing filtering based defense techniques available at browsers before reaching the server. • few complex exploits are hard to be detected by existing defense systems.
Bui et al. [8]	study XSS vulnerabilities in cloud-application add-ons.	• 9% of analyzed add-ons are found vulnerable to XSS. • a set of countermeasures are proposed to tackle the issue. These include: implement the add-on logic on the server instead of the client-side and prevent sharing access tokens to delegate all user permissions.
Talib and Doh [60]	evaluate and compare the performance of 12 dynamic open-source XSS filters.	• filters are not sufficient alone for the detection of XSS attacks. • tested filters work well with malicious scripts but not with benign ones (generate high false positives). • not suitable for the detection of DOM-based XSS (DXSS). • no guarantee that filters work properly with other contexts than HTML.
Korac et al. [61]	comparison of the impact of the three basic XSS attacks regarding a set of qualitative attributes demonstrating a topological relationship among them.	• the three basic XSS attacks can be combined to form hybrid attacks that are hard to be detected.
Shar et al. [62]	study the impact of integrating secure coding into undergraduate web programming courses.	• Through scanning a collection of projects established before and after integrating the cybersecurity course using the ZAP tool, the authors discovered a notable reduction in the number of XSS vulnerabilities detected in newly developed projects. Additionally, a conducted survey revealed that students emphasized the importance of the course in increasing their awareness of cybersecurity.

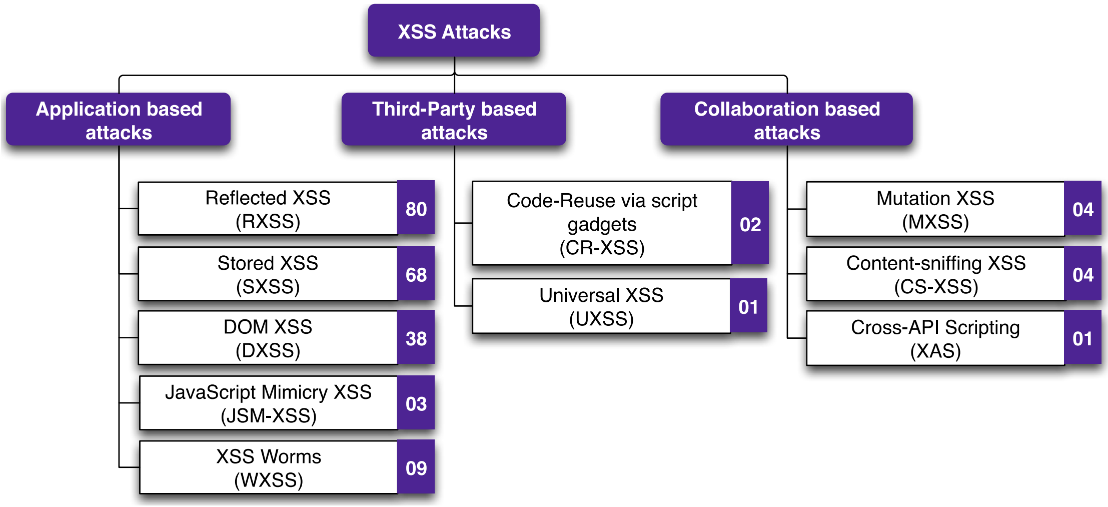
XSS Attacks
Application based attacks
Third-Party based attacks
Collaboration based attacks
Reflected XSS (RXSS) 80
Stored XSS (SXSS) 68
DOM XSS (DXSS) 38
JavaScript Mimicry XSS (JSM-XSS) 03
XSS Worms (WXSS) 09
Code-Reuse via script gadgets (CR-XSS) 02
Universal XSS (UXSS) 01
Mutation XSS (MXSS) 04
Content-sniffing XSS (CS-XSS) 04
Cross-API Scripting (XAS) 01

Fig. 9. Types of XSS attacks.

3. *DOM XSS (DXSS)*: DXSS attacks are render-time attacks. Contrary to RXSS and SXSS, malicious data used in DXSS attacks are used to dynamically alter the DOM trees generated by browsers at rendering phase. Attackers use DXSS attacks for web applications that accept data from user-controlled sources to be used as specific DOM objects values. The peculiarity of DXSS is that malicious data can be embedded in URLs as values to specific DOM objects or HTML elements but never reach the server. These attacks are also referred to in [63] as Type 0 and are firstly reported in 2005 by Amit Klein [66].
4. *JavaScript Mimicry XSS (JSM-XSS)*: Instead of injecting malicious scripts, attackers make use of scripts already used in web applications to launch JSM-XSS attacks. It has been reported and demonstrated with several examples in [67] that legitimate scripts injected in different places than those intended by developers may lead to harmful behaviors of web applications. JSM-XSS attacks are hard to be detected and easily pass whitelisting based filters.
5. *XSS worms (WXSS):* WXSS attacks are XSS attacks with self-replication capability. RXSS and SXSS attacks with self-replication capability are classified as WXSS. This distinction is adopted since specific other type of web application vulnerabilities are required for spreading attacks and become worms. WXSS attacks are more dangerous since they propagate among web application users and progressively infect other users over time and remain so until being detected and removed. Typical targets of such attacks are online social network applications where user profiles are somehow linked to each others. To launch a WXSS attack, web application vulnerabilities are used to infect the first user and inject a malicious script; the execution

of the injected script uses other application vulnerabilities to escalate the privilege and perform some actions on the user's behalf. Those actions enable self-replication and hence cause the infection of other linked users.

5.3.2. Third-party based XSS attacks

The attacks of this category do not directly rely on web application vulnerabilities, instead, they rely on third-party vulnerabilities such as browsers, browser extensions and third-party libraries or frameworks used by web applications. Two attacks found reported in the literature fall in this category:

1. *Code-Reuse via Script Gadget (CR-XSS)*: Triggering a CR-XSS attack basically requires a knowledge of the scripts included in the libraries and/or frameworks used by targeted applications. Those are also known as script gadgets. A successful CR-XSS attack is achieved by injecting HTML codes with disguised payloads (i.e., in non-executable form) that match DOM elements and provoke the execution of script gadgets. Apparently benign payloads are transformed into executable scripts by the execution of a single or chain of gadgets. CR-XSS is introduced in 2013 by Lekies et al. [42]. Script gadgets can also be sourced from user-land code, in this case, CR-XSS attacks become web application based-attacks.
2. *Universal XSS (UXSS)*: UXSS is caused by the lack of proper sanitizations of URLs by the browser itself or one of its extensions. For triggering a UXSS attack, intruders get benefits of vulnerabilities located in browsers or browser extensions. Therefore, by seducing users to click in a link that provokes the execution of installed plugins in the visitor browser, the plugin triggers the execution of malicious scripts incorporated in the links. They are called universal since malicious scripts can be executed in the context of any web site and not directly related to specific web applications.

5.3.3. Collaboration based XSS attacks

Attacks of this category can only be succeeded with the presence of web applications and third-party vulnerabilities. If any of those vulnerabilities is missing, those attacks fail. Three of the reported attacks in the literature fall in this category:

1. *Mutation XSS (MXSS)*: MXSS attacks are mainly caused by the capability of browsers to transform formatted HTML strings into valid DOM elements making use of the `innerHTML` property used in web applications. Attackers supply their malicious data as formatted HTML strings associated as values to the `innerHTML` property in vulnerable target web applications. Those malicious data may bypass application filters and transformed by browsers into valid HTML contents, they are inserted as new DOM elements and then executed as part of rendered pages. Typical targets of MXSS are webmail applications with HTML content message transmission capabilities. Transmitted messages are transformed (i.e., mutated) into valid scripts executed by the receiving browsers at rendering the content of messages. MXSS attacks are firstly reported in 2017 by Heiderich et al. [35].
2. *Cross-API Scripting (XAS)*: XAS is a kind of XSS attacks that targets web applications providing Restful APIs to third-party developers such as social networks. Attackers inject malicious scripts in their own application profiles and hence users of third-party applications become vulnerable to XAS. Malicious data are transmitted to victims' browsers through using APIs to retrieve data from web applications. The root cause of XAS attacks is the lack of proper sanitization of data by: (1) the web application itself that permits attackers to inject malicious data, (2) third-party web applications for their acceptance of malicious API responses without proper sanitization. This causes the execution of malicious data at the victims' browsers. The term XAS is firstly introduced in 2013 and later on 2015 by Zhang et al. [39].

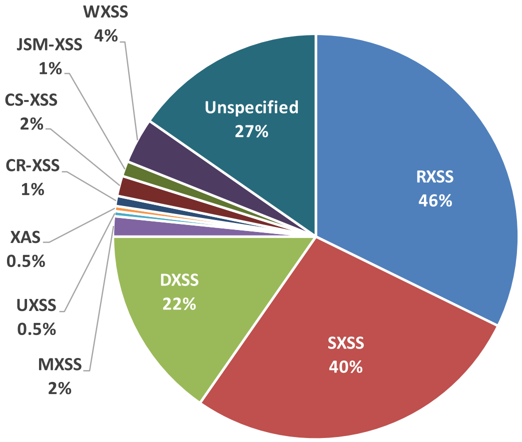

Fig. 10. Distribution of included studies per addressed XSS attack type.

3. *Content-Sniffing XSS (CS-XSS)*: XSS attacks caused by the misinterpretation of file content types by browsers are named content-sniffing XSS and referred to in this paper as CS-XSS. To conduct a CS-XSS attack, malicious data are supplied as part of separate media files (e.g., PDFs, images) uploaded to vulnerable web applications. Those files become harmful to every user loading them in a vulnerable browser. To be successful, CS-XSS attacks require web applications permitting the upload of infected files and browsers using a content-sniffing practice to deduce the type of files. With the presence of scripts in files, browsers consider them as HTML files and hence get rendered where the injected scripts are executed at the victims' browsers.

Fig. 10 shows the distribution of included studies regarding the different XSS attack types. Application-based XSS attacks are the most addressed, they are reported and handled by 98 studies individually, together or with other attack types. Specifically, RXSS is the most referred attack followed by SXSS and DXSS. The focus on RXSS is understandable since it is the most common and the easiest to be detected compared with other attacks. Remarkably, DXSS is getting much attention compared with the results reported in 2015 by Hydara et al. [18]. The other attacks are less addressed. Specifically, XAS and UXSS are only addressed once compared with other variants. The problem is due to the fact that those attacks target specific applications or platforms. XAS attacks target online social network applications with Restful APIs and UXSS targets specific browsers and browser plugins. Through the data extraction process, we found that studies with unspecified attacks are mostly referring to basic attacks, specifically RXSS and/or SXSS, however to be concise, we separated them and we did not consider them as basic attacks.

Fig. 11 shows how many studies with a specific research type are addressing each XSS attack type. The figure presents a two-x-y scatterplot with bubbles in the intersections. The size of a bubble is proportional to the number of studies that are in the pair of classes corresponding to the bubble coordinates (i.e., research type and XSS attack type). The sparse distribution of studies noticed in Fig. 11 indicates a bias toward basic attacks (i.e., RXSS, SXSS and DXSS). Other XSS attack variants are rarely studied. Specifically, there is a noticeable research gap regarding the prevention of such attacks.

Finally, Fig. 12 shows the distribution of studies regarding XSS attack languages. JavaScript is the dominant language for XSS attacks. Other languages such as AJAX and Flash are barely studied. This shows a bias toward JavaScript based XSS attacks. It is important to note that Fig. 12 only represents the languages used to develop XSS attack vectors, not the languages associated with web applications exposing XSS vulnerabilities.

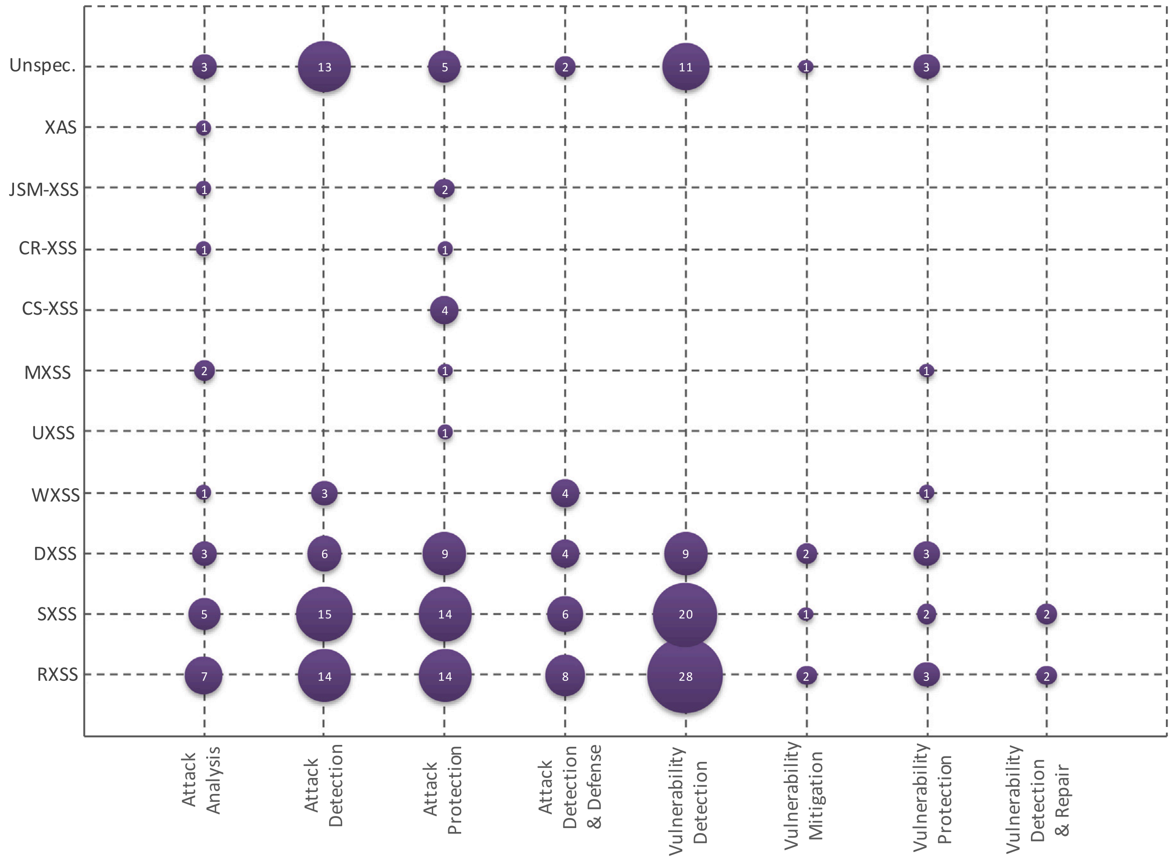

Fig. 11. Distribution of the research focus of included studies regarding the different type of XSS attacks.

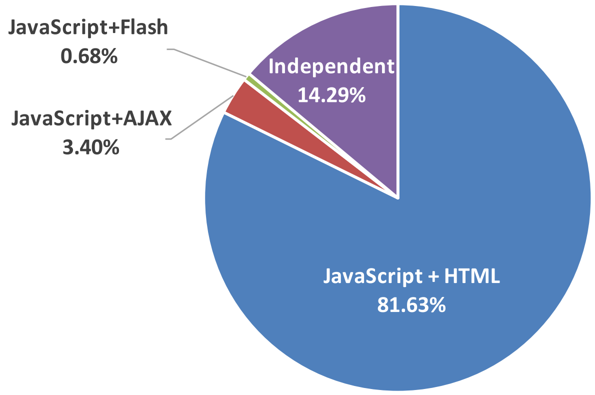

Fig. 12. Interest to different XSS attack languages.

5.4. What techniques have been used to tackle the XSS issue? (RQ4)

Different techniques have been proposed for handling XSS attacks. Those techniques can be categorized following their intents and natures. Fig. 13 gives an overview of the proposed classification of solution proposals together with the number of included papers addressing each technique.

5.4.1. XSS prevention techniques

Prevention techniques refer to the set of security measures that prohibit the appearance of attacks in the first place and enable the development of XSS-free web applications. Those measures need to be taken at early stages of the development process for Web applications, specifically, at the construction phase and prior deployment. They can be divided into XSS vulnerability mitigation, detection and/or defense techniques.

XSS vulnerability mitigation techniques

XSS attacks are mainly caused due to the absence of proper sanitization of user-controlled inputs to web applications. In order to reduce the chances of being attacked, several measures can be taken by web application developers. The study of the literature shows three main secure coding practices: use of *secure APIs*, use of *secure web templating* and instrumentation of *proper sanitizer routines*. In the sequel, we provide a brief description of each proposal and discuss its limitations.

Use of secure APIs:. Browsers often come with built-in APIs that enable performing complex operations, such as manipulating the DOM structure of web documents, in a high-level manner. Those APIs are not without flaws; they can be used by intruders to launch DXSS attacks. Wang et al. [43] proposed developing secure web applications by forbidding direct calls to vulnerable JavaScript and TypeScript engines. Instead, secure APIs need to be developed, integrated and made available for web developers. The new APIs enforce typing of HTML element values to prevent any future script injection. However, new APIs also inherit the unsoundness of JavaScript compilers. In addition, learning new APIs requires much efforts from developers which is a non cost-effective solution for large development organizations and workflows. Musch et al. [65] proposed a lightweight solution that consists of using DOM API wrappers to prevent the injection of scripts through calling built-in DOM API functions. Occasionally, if values to DOM APIs are required from third parties, those values are redirected to an API wrapper for sanitization before being forwarded to standard APIs. Unfortunately, the proposed scheme is unable to wrap all JavaScript sinks such as *eval()* and has a compatibility issue to be automatically integrated to all web applications. Consequently, users need to be interfered to manually add wrapper scripts at the top of each web page which is a tedious task.

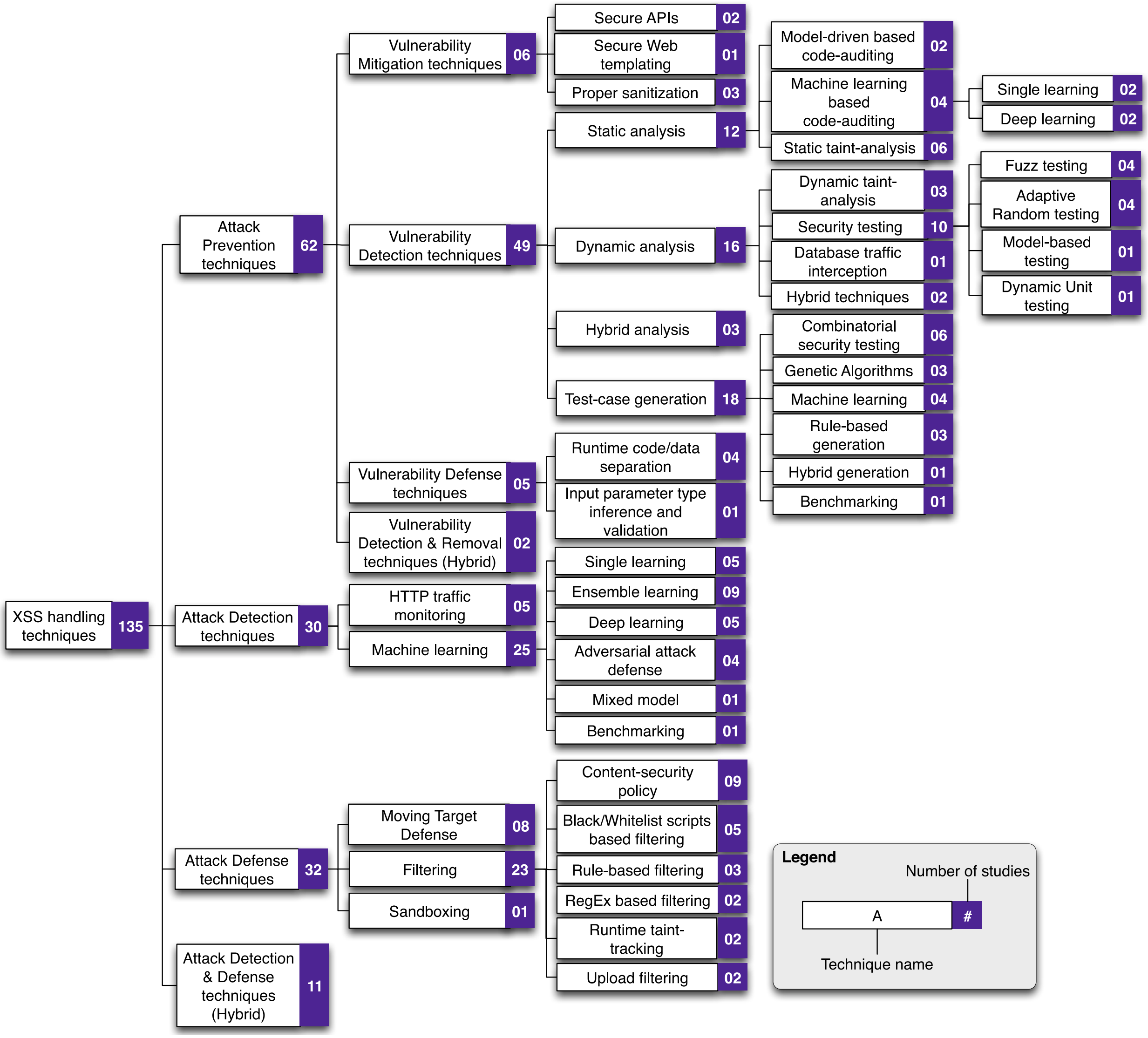
XSS handling techniques 135
Attack Prevention techniques 62
Vulnerability Mitigation techniques 06
Secure APIs 02
Secure Web templating 01
Proper sanitization 03
Vulnerability Detection techniques 49
Static analysis 12
Model-driven based code-auditing 02
Machine learning based code-auditing 04
Single learning 02
Deep learning 02
Static taint-analysis 06
Dynamic analysis 16
Dynamic taint-analysis 03
Security testing 10
Fuzz testing 04
Adaptive Random testing 04
Model-based testing 01
Dynamic Unit testing 01
Database traffic interception 01
Hybrid techniques 02
Hybrid analysis 03
Test-case generation 18
Combinatorial security testing 06
Genetic Algorithms 03
Machine learning 04
Rule-based generation 03
Hybrid generation 01
Benchmarking 01
Vulnerability Defense techniques 05
Runtime code/data separation 04
Input parameter type inference and validation 01
Vulnerability Detection & Removal techniques (Hybrid) 02
Attack Detection techniques 30
HTTP traffic monitoring 05
Machine learning 25
Single learning 05
Ensemble learning 09
Deep learning 05
Adversarial attack defense 04
Mixed model 01
Benchmarking 01
Attack Defense techniques 32
Moving Target Defense 08
Filtering 23
Sandboxing 01
Content-security policy 09
Black/Whitelist scripts based filtering 05
Rule-based filtering 03
RegEx based filtering 02
Runtime taint-tracking 02
Upload filtering 02
Attack Detection & Defense techniques (Hybrid) 11
Legend
Number of studies
A
#
Technique name

Fig. 13. Taxonomy of proposed solutions to XSS attacks.

***Use of secure web templating:*.** The use of Web templating is a common practice that enables web designers and developers to generate their customized web applications through reconfigurations of already designed templates. Samuel et al. [68] proposed the construction of secure web applications making use of web templating frameworks with embedded auto-sanitization mechanisms. The proposed system parses templates, infers input variable types and annotates them with context type qualifiers. Annotated templates are then compiled into the target language of pages with instrumented sanitizers from a sanitizer library. The solution is experimented only with Google Closure framework templates and unable to determine all variable contexts due to browser quirks.

***Use of proper sanitization routines:*.** Instrumenting sanitizers at all sensitive injection points is an intuitive solution to prevent XSS attacks. Huang et al. [69] proposed a sanitization tool named WebSSARI. The proposed tool can be used by web developers to perform automatic static taint-tracking; tainted data reaching sensitive sinks are treated by automatic integration of sanitizers from a database. The reliance to a predefined set of sanitizers is the main drawback of the approach and the experiments conducted by the authors showed that the tool produces large number of false positives (26.9%). Heiderich et al. [70] proposed a ready to use sanitizer named DOMPurify. It enables transforming malicious HTML strings into safer versions by striping out every dangerous and confusing characters and thereby prevent XSS attacks. It also prevents bypassing through encryption where scripts are firstly decrypted and filtered prior rendering. However, DOMPurify can easily be passed by CR-XSS attack vectors. In addition, DOMPurify need to instrumented manually at each injection point which is not feasible in practice. To enrich user-libraries with correct and effective sanitizers, Hooimeijer et al. [71] proposed a domain specific language (BEK) to describe the behavior of sanitizers. BEK descriptions can be transformed into a special finite state machines to be formally checked for their correctness and effectiveness against XSS attacks. BEK can detect some formal properties of single and composed sanitizers such as commutativity and idempotence. Correct sanitizers can automatically be transformed into Java or C# for being integrated into existing systems. Unfortunately, tested santizers need first to be transformed manually to BEK which is a tedious and error-prone task.

XSS vulnerability detection techniques

Missing flaws by secure-coding practices can be detected using appropriate approaches and tools for scanning web applications prior deployment. Vulnerability detection techniques aim at identifying potential injection points with no proper validation by examining the source code or the behavior of web applications. When identified, the developers are notified of their presence. They become responsible for providing the necessary defense techniques. The literature is rich with vulnerability detection techniques that can be classified into four main categories: *static*, *dynamic*, *hybrid analysis* and *test-case generation*.

Static analysis. Static analysis approaches and tools focus on exploring web application source codes, without their execution, to find security vulnerabilities. They have the ability to discover all the execution paths which effectively reduces the false negative rate. Three main static analysis approaches are proposed by included studies: *model-driven based*, *machine-learning based* code auditing and *static taint-analysis*:

- **Model-driven based code auditing:** web applications vulnerabilities can statically be checked through the analysis of their derived abstract models. This technique is explored by two included studies [72,73]. In [72], the defense measures instrumented by developers are recovered and checked for correctness and consistency. For this sake, web application pages are transformed into control flow graphs. Nodes of generated graphs are classified based on security measures instrumented by developers on each node. A set of well-defined rules are checked to verify the adequacy of used defense measures in preventing XSS attacks on those nodes. The solution requires manual specifications of escaping APIs used in web applications; this makes it unsuitable for large web applications. In [73], the behavior of web applications is modeled using a well-defined domain specific language (DSL) making use of functional specifications of web applications. Test-patterns are explicitly described using natural language and automatically transformed into formalized forms named test-purposes. Test-cases are then generated from test-purposes and executed on the abstract behavioral model for verification. This solution is also not suitable for large applications since it requires user interventions for the description of test models using the proposed DSL.
- **Machine learning based code auditing:** taking benefits from already discovered and known vulnerabilities, machine learning can be used as a code auditing approach to check the vulnerability of developed web pages against XSS attacks. The approach is found being explored by four included studies [74–77]. Datasets constituted of safe and vulnerable web pages are collected from different sources such as GitHub[3] and Common Weakness Enumeration (CWE).[4] Features are extracted from web page sources through feature engineering like in [74–76] or through feature learning like in [77]. Feature vectors are built and fed to machine learning classifiers for training and then for classifying web pages into vulnerable and safe. Two types of machine learning are found used in the literature: single learning in [74,75] and deep learning in [76,77]. Specifically, a variant of decision tree classifier (J48) is used in [78], Multi-layer Perceptron classifier (MLP) is used in [74,77] and Bidirectional Long short-term memory network (BLSTM) is used in [76]. Those solutions are language dependent (PHP) and produce high false positive rates.
- **Static taint-analysis:** consists of tracking input data from sources to sensitive sinks in web application source codes without being executed. An XSS vulnerability is reported if an input data reaches a sensitive sink without being properly sanitized. Jovanovic et al. [79] developed a typical tool named Pixy that enables static taint-analysis through exploring automatically generated control-flow graphs. Pixy is designed for the detection of RXSS vulnerabilities in PHP web applications. Wang et al. [80] extended Pixy for the support of SXSS vulnerabilities. The proposed algorithm in [80] uses both data-dependence and control-dependence graphs to identify program slices causing SXSS vulnerabilities. However, Pixy is a language dependent tool and does not support PHP object-oriented features. In addition, it produces high false positive rates (around 50%) and is unable to detect flaws across multiple pages. Steinhauser and Gauthier [81] developed JSPChecker which is an another tool implementing static taint-analysis for the detection of XSS vulnerabilities. Instead of tracking input data to sinks, JSPChecker tracks the list of sanitizer routines applied to input data before reaching sinks in order to discover context inconsistencies. The solution requires manual identifications of sanitizers and mapping them to safe output contexts. Wassermann and Su [82] used static taint-analysis to check the ability of vulnerable inputs sent by users to invoke the JavaScript interpreter. The solution is browser dependent and cannot handle arbitrarily complex and dynamic codes. Yan and Qiao [83] proposed an optimized algorithm for static taint-tracking through reverse code auditing. Instead of exploring all existing paths, only few and relevant paths are need to be explored for the detection of vulnerabilities. However, insufficient experiments are conducted to check the correctness of the proposed algorithm. Su et al. [84] have proposed a novel approach for detecting vulnerable paths in PHP web applications, using a sanitizer-based tracking mechanism. The proposed approach is based on the hypothesis that the existence of a sanitizer in a specific location in a program indicates the arrival of untrustworthy input, which will subsequently be used in security-sensitive points of the same program. The process of the proposed approach involves generating a data-flow graph of the application and conducting a sanitizer-based identification, encompassing both default and user-defined sanitizers. The detected sanitizers are then used to infer potential inner sources and sinks through a comprehensive backward and forward analysis of the data-flow graph. This approach facilitates the detection of properly sanitized paths as well as those that have been overlooked. To achieve this aim, the proposed approach combines data-flow analysis and natural language processing to identify sanitizer functions, along with taint-analysis techniques for detecting vulnerable paths within the application. Despite the demonstrated efficiency of the proposed technique, the approach does have limitations. One limitation is the detection of all user-defined sanitizers, particularly those not annotated with comments or not wrapped around known ones. Additionally, another limitation lies in the dependence on applied sanitizers.

Dynamic analysis. Contrary to static analysis, dynamic analysis approaches and tools aim at discovering flaws of web applications while they are running. They enable testing web applications on real-time scenarios. Dynamic analysis methods focus on information acquired at runtime. They detect the presence of an XSS vulnerability based on HTTP responses by dynamically sending requests to servers. Dynamic analysis methods do not rely on the presence of web application codes and have generally lower false positive rates. Several dynamic techniques are used in the literature. In the sequel, we describe each of those approaches and we discuss the correspondent studies included in this review.

- **Dynamic taint-analysis:** consists of marking input sensitive data and track their propagation when tested applications are running. A vulnerability is reported when input data reach predefined program locations or sinks. Martin and Lam [85] proposed a taint-analysis technique armed with a model checker to prove the success of attacks. The proposed system is designed to handle taint-based vulnerabilities including XSS and SQL injections. Tested vulnerabilities are described using a specific language named PQL [86]. Users should provide the PQL specifications of tested vulnerabilities together with a set of input query parameter values. The proposed system explores the space of possible input tests and monitors the execution of the application under each

[3] GitHub home page: https://github.com

[4] CWE: Common Weakness Enumeration: https://cwe.mitre.org

input. The model checker checks whether input tests successfully reach specified sinks and reports the correspondent attack paths. The efficiency of this solution relies on the correctness of specifications given by developers. It fails at the detection of vulnerabilities with the presence of advanced sanitizers and it is only applicable to Java based web applications. Steinhauser and Tuma [87] developed a tool that uses an extended dynamic taint-tracker for the detection of browser-context-sensitive XSS flaws. They are vulnerabilities caused by the use of inappropriate sanitizers for particular browser contexts. The proposed tool performs a dynamic taint-tracking to identify the set of sanitizers applied to tainted values. Afterwards, a model browser checks the resulted HTML responses to determine the browser contexts where tainted data are passed through. A set of well-defined rules are used to check the compatibility of used sanitizers with identified browser contexts. The proposal is not suitable for the detection of DOM-based vulnerabilities and is applicable only to Django based web applications.

Instead of monitoring program execution, taint-inference matches input values from HTTP requests with those resulted from their correspondent HTTP responses and infers the taint-flow between them. Any match is reported as an XSS vulnerability. Pan et al. [88] proposed a new taint-inference technique for the detection of web applications vulnerability to RXSS attacks. To overcome the effects of URL rewriting and HTML sanitization on matching results, the authors used local sequence alignment and removal gap penalty inspired from molecule sequence alignment used in bio-informatics [89]. The technique performs poorly when tags are removed by HTML sanitizers.

- **Security testing:** Security testing refers to the set of techniques providing evidences that web applications are safe and reliable, and that they do not accept unauthorized inputs. They test the impact of malicious or unexpected inputs on their functionalities when they are running. Several techniques have been used for the detection of XSS vulnerabilities:

  - **Fuzz testing:** consists of discovering web application vulnerabilities by injection of invalid, malformed, or unexpected inputs and observation for exceptions on their behaviors. This technique is explored by four included studies [90–93]. McAllister et al. [90] proposed a guided fuzzer by a collected set of use-cases from the server. The fuzzer (w3af) replaces the input parameter values of real-world requests with malicious strings from a database. The solution is only experimented with Django web applications. Duchène et al. [91] proposed the use of a guided w3af fuzzer with an inferred control and data-flow model to increase the capability of detecting XSS vulnerabilities. The solution does not support all encoding functions and requires resetting repeatedly tested applications to their initial states. In a recent work [92], the authors replaced the w3af fuzzer with more appropriate one that uses well-defined attack grammars to transform benign inputs into fuzzed inputs. The new solution requires the intervention of users to identify non-deterministic values of specific attributes related to the implemented tool. Eriksson et al. [93] proposed a black-box scanner that loads pages in a modified browser and observes their execution flows. Fuzzed inputs are sent and the scanner detects whether they have been executed. The solution fails to detect vulnerabilities when too much randomness is used on inputs.
  - **Adaptive random testing:** instead of testing web applications using arbitrary set of test-cases, adaptive random testing can be adopted to select more distinctive and likely successful test-cases for each step [94]. Lv et al. [95] adopted a distance based selection of test-cases. Rocha and Souto [96] proposed to test web applications on generated test-cases according to contexts and qualifiers associated to each discovered vulnerable point. Leithner et al. [44] proposed a feedback based approach for the selection of next injected test-cases. Therefore, based on the current injected vector and the reached output context, the next attack vector is an update of the current vector that likely reaches a number of defined ideal locations making use of weights. Those weights are required to be selected by developers which may considerably affect the performance of the proposed system. Tripp et al. [97] used reinforcement learning for the selection of likely successful attack vectors. Using a database containing 500 million payloads and through a learning capability from failed attacks and previously selected payloads, the algorithm selects, in each step, a payload that likely bypass instrumented sanitizers in web applications. The solution produces false negatives with the presence of sanitizers that limit input lengths or trim a fixed number of characters from the head or the tail of input strings.
  - **Model based testing:** in such kind of techniques, trained models are used to distinguish vulnerable from non vulnerable pages. Avancini and Ceccato [98] developed a test-oracle named Circe for testing XSS vulnerabilities of web applications. As a test oracle, a model is constructed based on the structure of web pages generated using safe inputs. Malicious inputs from a library are injected and resulted page structures are matched to those generated from safe inputs. Any deviation is reported as a potential vulnerability. The model is unable to detect all real-world scenarios due to the adopted coverage criterion and its reliance to a fixed set of malicious inputs.
  - **Dynamic unit testing:** unit testing is another dynamic analysis technique used for vulnerability detection. In dynamic unit testing, a program unit is executed in isolation with selected inputs, the results are then compared with the expected outcomes. Mohammadi et al. [99] proposed the use of a dynamic unit testing approach for the detection of XSS vulnerabilities in JSP web applications. Unit tests for each page are automatically constructed and executed using a unit test execution framework. For test inputs, a well-established attack grammar is used for the generation of proper inputs.

- **Database traffic interception:** providing a control interface between web applications and their databases enables XSS analyzers to detect SXSS vulnerabilities. Steinhauser and Tuma [100] proposed extending existing black-box scanners with a database response payload injection mechanism. For this sake, the authors designed a database traffic interception protocol that records the activity of web application databases and injects exploited patterns. Accordingly, the database reports to the analyzer every injection of a data string. Regular expressions are used to match injected patterns in received responses. The analysis process was much slower compared with ordinary black-box scanners.
- **Hybrid dynamic analysis:** hybrid dynamic analysis combines two or more dynamic analysis techniques for the detection of injection vulnerabilities. Two hybrid solutions have been found in the literature [101,102]. Melicher et al. [101] proposed and experimented the use of a deep learning model as a pre-filter for a dynamic taint-tracker. A deep learner model is used to check the vulnerability of JavaScript codes, only unconfirmed malicious samples are subject to a dynamic taint-tracker for analysis. A deep learner is trained on a collected dataset of confirmed and unconfirmed malicious scripts. Dynamic taint-tracking is performed making use of a modified browser with embedded taint-track analyzer. The solution is browser dependent and produces high false

Table 7
Adopted combinatorial security testing models.

Study	#parameters	Best *t* value	#generated cases	Attacks
Bozic et al. [105]	11	4	8761	RXSS, SXSS
Simos and Kleine [106]	7	4	6891	RXSS
Garn et al. [107]	7	3	7200	RXSS
Simos et al. [104]	5, 4, 3	3	149, 114, 27	RXSS
Garn et al. [108]	12	2	99	Unspecified
Leithner et al. [44]	2	2	42	RXSS

positives. Ayeni et al. [102] proposed a black-box scanner with embedded fuzzy inference system. A vulnerability is detected if one of seven DOM-based API features contains unsafe data given as input in a URL. The values of DOM features are extracted form HTTP responses and subjected to a fuzzification process. 21 rules are used by the inference engine to produce a fuzzified output which is deffuzified and interpreted as a presence or absence of a vulnerability. More experiments are needed to confirm the suitability of the proposal with large web applications.

Test-case generation. Some static analysis based approaches, such as taint-analysis, and all dynamic analysis based approaches require a collection of input data to perform their assessment processes. The collection should include various, distinct and representative input tests to cover all the execution paths or scenarios and reveal potential fails. For systematic generation of input tests, several techniques are explored:

- **Combinatorial security testing:** test inputs are designed as a model named input parameter model (IPM). The model is described as a context free grammar constituted of a finite number of parameters, each of which can take one of a finite set of values. A valid input test is obtained through *t-wise* combinations of input parameter values with a specific value *t*. Constraints are useful means to reduce the input space and enforce the generation of valid, more comprehensive and sophisticated test inputs [103]. Six studies adopted such technique for the generation of successful XSS attack vectors. Table 7 describes the details of the models proposed in each study together with the targeted attacks. Specifically, Simos et al. [104] proposed three sub-grammars, each targets a specific HTML context.
- **Genetic algorithms:** in the aim to find the minimal number of test cases that reveal as many XSS vulnerabilities as possible, three studies combined static taint-analysis and genetic algorithms (GA). Firstly, a static taint-analyzer is used to identify vulnerable paths in the code of web applications. Then, a GA is used to generate test inputs to be injected and confirm the vulnerability of those paths. The algorithm starts with an initial collected set of candidate solutions, computes the fitness value of each candidate, selects candidates based on computed fitness values and generates new candidates by altering selected ones making use of crossover and mutation genetic operators. This white-box based technique enables understanding web application vulnerabilities before fixing them, but also inherits the limitations of static analysis (i.e., high false positives). Ahmed and Ali [109] and Marashdih et al. [110] adopted a similar approach. Pixy tool [79] is used to extract vulnerable paths and a genetic algorithm is used to confirm their vulnerabilities. The only difference is that in [110], the process was optimized through removing infeasible paths from the test process. The proposed solution produces less false positives but infeasible paths are require to be removed manually. Avancini and Ceccato [111] proposed a new improvement where concrete symbolic execution [112] is used to avoid local optimum caused by the GA. Specifically, a local search based on constraint solver is used to minimize the local optimum effect of the GA by the selection of appropriate input values that traverse more target paths. This approach is repeated whenever a local optimum is found by the GA. Liu et al. [113] proposed an optimized genetic algorithm to generate effective XSS vectors. The approach begins with testing the target web application using randomly generated test data to identify the locations where the test data appears. With this information and considering a well-defined grammatical structure of XSS attack vectors, an initial set of individuals is generated. Each initial individual undergoes fuzzing on the web application, and its fitness is evaluated based on the feedback received. Crossover and mutation operations are then applied to the individuals. Mutation involves using well-defined operations, including the random selection of different encoding patterns for the attack vectors. The process is repeated, performing fuzzing tests and fitness calculations, until the attack vectors are successfully executed or reaching a specified number of iterations. While the experiment shows a reasonable performance on the detection of XSS vulnerabilities with acceptable time efficiency, the process involves manual testing of generated attack vectors.
- **Machine learning:** machine learning are also used to generate valid attack vectors for the detection of XSS vulnerabilities. Caturano et al. [114] proposed a reinforcement learning based approach. An attack string is divided into five distinct sections. A reinforcement learning agent acts by modifying only one of those sections at each step. It is trained with the help of the user to reach a valid attack string using standard Q-learning [115]. This approach has higher accuracy and low false positives compared with other automated scanners but requires an extensive developer intervention in the training process. Frempong et al. [116] proposed a tool that uses machine translations (encoder/decoder) with part-of-speech tagging (POS tagging) to generate JavaScript exploits from intents specified in natural languages. After training, the model becomes capable of generating executable benign and malicious samples to be tested in real-world applications. However, the model generates only exploits related to specified intents. Foley and Maffeis [117] proposed a framework for detecting XSS vulnerabilities using deep reinforcement learning (DQN). The framework incorporates two types of agents: a single escape agent responsible for executing escape operations, and 18 sanitization agents to perform mutation operations to bypass sanitization routines. These agents collaborate in a two-step process where the escape agent starts, and if a sanitization routine is detected through response analysis, the attack vector is redirected to the sanitization agents. The conducted experiments demonstrated improved detection compared to seven well-known black-box scanners, enabling the detection of new vulnerabilities. However, the proposed framework incorporates a weak crawler, which may fail to detect all injection points, leading to higher false positives and false negatives. Additionally, the agents use an optimized number of mutation actions, reducing the diversity of generated payloads. Furthermore, the study lacks runtime analysis, which is crucial for black-box scanning frameworks and tools. Lee et al. [118] introduced a framework for detecting RXSS vulnerabilities in online web applications, using the advanced actor-critic reinforcement learning algorithm. Within the proposed framework, a single agent is responsible for generating and mutating attack vectors, incorporating seven actions of generation and 32 actions of mutation. The authors built a prototype on top of the Wapiti black-box scanner, which demonstrated superior performance compared to existing black-box scanners, all while maintaining a reasonable response time. However, it is important to note that the framework's performance heavily relies on the Wapiti crawler for identifying and prioritizing injection points. Additionally, the limited number of generation and mutation actions hinders the diversity of generated attack vectors.

- **Rule based generation:** rules enforced by taint-tracking are also used for the dynamic generation of test-cases. Lekies et al. [119] modified the Chromium open source browser by integrating byte-level taint-tracking mechanism into its embedded components. The taint-tracker identifies source, sink and applied built-in filters to each tainted value in data flows. Those information are used by a control backend to generate valid XSS exploits appended to URLs to avoid embedded filtering capabilities of browsers. The generation is performed making use of well-defined rules associated to each source (HTML tags, nodes, comments or Javascipt). The technique is fully automated and enables the generation of exploits for the detection of DXSS vulnerabilities. Bensalim et al. [120] improved the generation rules adopted in [119] by specifying URL positions where injected exploits are most likely to result in successful attacks. This enabled the detection of 1.9% more vulnerabilities. The approach is limited to direct paths from sources to sinks with disabled URL-encoding which makes it less effective against complex attacks and real-world scenarios. Wang et al. [121] proposed a system that receives URLs and preprocesses them, loads their correspondent pages and obtains taint-traces. Those are used to automatically generate attack vectors through well-defined rules and verify their effects. The approach enabled the detection of 1.8% more vulnerabilities than the commercial AWVS vulnerability scanner[5]. However, the technique is time expensive when it comes to generate complex attack vectors.
- **Hybrid based test-case generation:** Kiezun et al. [122] proposed a hybrid and automatic generation technique of executable input tests for both XSS and SQL injection vulnerabilities. The process starts with automatically generated inputs by Apollo [123], an input generator based on concrete and symbolic execution [112]. Those inputs are injected into web applications and checked if they reach sensitive sinks. If it is the case, inputs are mutated making use of an attack pattern library. Besides RXSS, the technique enables the detection of SXSS vulnerabilities through tracking flows of tainted data in web application databases. However, the proposed technique does not consider attacks across multiple pages and it is language based solution (PHP/SQL web applications).
- **Benchmarking:** benchmarks of test-cases are indispensable for the development of robust models and fair evaluation of different XSS detection methods. Pan and Mao [124] proposed a micro-benchmark containing 175 test cases. Those test-cases are automatically generated from a template. The template is an abstraction of a typical vulnerable HTML page to DXSS attacks, it is constituted of six elements: source, propagation, transformation, sink, trigger and context. Associating values to each of those properties resulted on an executable attack vector. Enlarging the values of those properties enables the generation of more test-cases. However, the proposed benchmark does not cover complex attacks caused by language features and browser quirks.

Hybrid analysis. Hybrid analysis consists of using static and dynamic analysis in combination or alternation. Besides the detection of all data-flow paths, it is intended to produce low false positive rates. Pan and Mao [125] proposed an alternation of static and dynamic analysis for the detection of DXSS vulnerabilities caused by the Greasemonkey browser extension [126]. Greasemonkey is a cross platform extension that enables users to write personalized JavaScripts to customize web page appearances and behaviors. In the static analysis phase, user scripts from of the Greasemonkey extension are filtered by matching patterns related to specific privilege granting directives. In addition, a static parser is used to identify source–sink paths and enable the elimination of safe contents. Remaining scripts are subjected to a dynamic analysis (i.e., symbolic execution) to confirm their vulnerabilities. The approach has large overhead and requires manual assistance for the generation of suspicious event sequences. Van Acker et al. [127] used a combination of static and dynamic analysis for the detection of rich Internet application vulnerabilities to XSS. Static analysis is used to automatically identify the set of ActionScript sensitive variables used in files that can be affected with user inputs. This step is performed through de-compiling SWF files and using regular expression matching. Dynamic analysis is used to test identified variables in the aim to discover vulnerabilities. Malicious payloads are generated making use of 10 templates and injected as values to sensitive variables and checked for malicious effects. The approach fails to cover all vulnerabilities due to the adoption of an automatic clicker simulator which resulted on high portion of false positives. Balzarotti et al. [128] combines static and dynamic analysis for testing the effectiveness of used sanitization. Static taint-analysis is used to identify applied sanitizers in web applications. Dynamic analysis is used to check the effectiveness of sanitizers by injection and tracking malicious inputs. If sensitive sinks are reached, sanitizers in the correspondent path are reported as ineffective.

XSS vulnerability defense techniques

In order to protect web applications from XSS vulnerabilities at runtime, two major techniques are proposed: *runtime data-code separation* and *input parameter type inference and validation*. Vulnerability defense techniques stop XSS assaults before they ever start. They reinforce input data control and prohibit the execution of harmful user inputs at runtime.

Runtime data-code separation. In order to prevent the execution of malicious scripts already injected into vulnerable web applications, scripts at sensitive points of HTTP responses can be extracted, at the server-side, and sent separately in a safe manner to the client-side in response to each HTTP request. If handled appropriately, malicious scripts can be prevented from being executed by web browsers. In this context, Louw and Venkatakrishnan [129] proposed a code-data separation method that guarantees the safe construction of HTML parse trees on the web browser. Special codes are instrumented into sensitive points of web applications to call the server whenever reached. In response to each call, the parse tree of data at each point is prepared at the server-side and transmitted as string literals to prevent its execution by the browser. This releases browsers from handling suspicious scripts. In this proposal, developers are asked to manually annotate code positions holding untrusted data which is impractical for large web applications. Inspired from SQL binding mechanism, Iha and Doi [130] proposed a similar technique for XSS. Web applications and browsers are modified to communicate a user modified agent request header. HTTP responses will then include the HTML structure without any parameter value. Those are included, following a specified syntax, in the HTTP response header. Consequently, the browser generates the DOM tree with empty parameter values and then uses the binding values from the HTTP response header and replaces them as literals that cannot be executed by browsers. Unfortunately, the proposed scheme does not support complex HTML structures, events and style attributes. Parameshwaran et al. [131] used dynamic taint-analysis to infer benign DOM tree templates that can be generated with the presence of malicious user inputs. Those templates are stored into a database which is used at runtime to decide the best template fitting the runtime input. The experiment showed that the solution has a reasonable overhead but requires major changes in the code of web applications as well as browsers. Gupta et al. [45] proposed a different approach. For every HTTP request, the server generates the correspondent HTTP response, extracts and isolates scripts in separate files and modifies the response code accordingly. At the client-side, extracted scripts are analyzed using

[5] Acunetix web vulnerability scanner: https://www.acunetix.com/plp/web-vulnerability-scanner/

taint-analysis and untrusted variables are identified. The output string of each untrusted variable is checked. If no suspicious pattern matches, the HTTP response continues to the user, otherwise the parameter value of the HTTP request and the URI link, if any, are extracted and decoded. A similarity with stored suspicious variables is measured. If similarity found, a vulnerability is reported and users are redirected to safe or error pages. The authors showed that the approach is convincing in terms of detection of vulnerabilities but no performance overhead analysis is performed.

Input parameter type inference and validation. Most scripting languages used for developing complex web applications are untyped languages. This is exploited by attackers to inject scripts instead of clear values to input fields which leads to XSS attacks. Therefore, enforcing types on sensitive data values is an intuitive solution to protect against XSS attacks. In this context, Scholte et al. [132] proposed a system that infers and validates input parameter values to protect against XSS and SQL injection vulnerabilities. At the training phase, types of input parameter values are inferred from benign test inputs and stored into a database. At runtime, types of input parameter values included in HTTP requests are inferred. Received and stored types are compared; if no match is detected, HTTP requests are dropped. The technique is fully automated but it is unable to extract all possible parameters especially in the case of encrypted or encoded HTTP requests.

XSS vulnerability detection and removal techniques

For complete protection, studies proposed detection and repair techniques. Shar and Tan [133] used static taint-analysis from their previous work [72] to identify input and potentially vulnerable output nodes from the generated control-flow graphs of web application pages. Pattern matching is used to identify the HTML context of each node. Finally, adequate escaping mechanisms of OWASP are identified by matching the OWASP XSS prevention rules and instrumented making use of the ESAPI API[6]. Marashdih and Zaaba [46] proposed an approach for the detection and removal of RXSS and SXSS vulnerabilities from PHP web applications. Static taint-analysis of Pixy tool [79] is used to examine the source code of web application pages and generate their correspondent control-flow graphs. For optimization purpose, infeasible paths are removed from the resulted graphs. A GA is used to generate different test-cases from an initial population of XSS attack vectors. The fitness function is used to evaluate the traverse of each input to target paths; a path is considered vulnerable if the GA generator succeeds to traverse it with zero fitness value. For removal, the HTMLPurifier library[7] is used to sanitize user data at detected vulnerable paths. The solution is language dependent and, like the proposal of Shar and Tan [133], it is unable to repair vulnerabilities caused by information flows across multiple pages.

5.4.2. XSS attack detection techniques

Runtime detection of attacks forms the last defense line against new and unknown attacks in the web. It is needed to cover risks that vulnerabilities' analysis and repair fail to deal with. It can also be considered as an alternative protection solution when detection and repair of vulnerabilities become hard to be performed in certain circumstances such as the case of legacy, large and critical web applications. The techniques of this kind can be installed independently of web applications at the client, server, proxy or multiple sides to detect and report attacks when they occur. The present review identified two explored techniques for runtime attack detection: *HTTP traffic monitoring* and *machine learning.*

HTTP traffic monitoring

By runtime HTTP traffic monitoring, every request and response related to a running web application is captured and checked for potential alteration by a non-controlled user data. Five studies adopted such approach are summarized in Table 8. Each solution proposal is described in terms of adopted detection scheme, installation location, pros and cons.

Machine learning

Different machine learning based models have been built for the detection of XSS attacks. Those models are developed in the aim to learn hidden properties of attack vectors and make correct predictions at runtime. Twenty-three studies included in this review used machine learning to achieve this aim. Different types of machine learning have been explored: single, ensemble and deep learning. Table 9 summarizes the proposed models together with the type and number of features and the best obtained performance for each model. In the sequel we discuss only remarkable proposals:

Besides traditional features extracted from URL and HTML contents, Wang et al. [134] and Rathore et al. [135] experimented new kind of features named OSN for the detection of XSS worms (WXSS). The OSN features capture the observed behavior on targeted social networking services. They include the spreading speed and frequency of suspicious data in the network traffic. The authors experimented several classifiers with and without the OSN features and found that the new features are more discriminating.

Mereani and Howe [136] proposed a two stage based classification system. At the first stage, a decision tree classifier is used to predict the nature of user inputs (i;e., texts or scripts). Scripts are subjected to a second classification stage with an ensemble classifier to predict their maliciousness. In total, 62 alphanumeric and non-alphanumeric features are extracted and used for classification.

Zhang et al. [137] used two separate Gaussian mixture models (GMM) trained on normal and XSS payloads respectively. The produced probabilities of the two GMMs on tested payloads are compared to reach a final prediction. They found that the combination of features extracted from HTTP requests and responses increases the accuracy score of the model.

Li et al. [138] used Random Forest (RF) as a semi-supervised learner. To rectify mislabeled data, a weighted neighboring process (weighted KNN) is adopted to re-label samples regarding their weighted distances to other similar samples.

Zhou and Wang [139] used an ensemble learner formed by a voting of several Bayesian Networks. In the aim to improve the prediction of the proposed model, the predictions obtained from the network are combined with a set of extracted threat intelligence rules including malicious IP addresses and domains from PhishTank[8] and FireHOL[9].

In order to enforce the detection of machine learning based models against adversarial attacks, several evolutionary techniques are proposed for the generation of adversarial attacks from regular attacks. Most of these techniques used reinforcement learning to achieve this goal:

Fang et al. [140] proposed a system that incorporates a detection model and an adversarial attack model. XSS payloads detected by the detection model are subjected to a set of disguising actions performed by an agent based on a Double Deep Q-Network algorithm (DDQN). Those actions include encoding, obfuscation, sensitive word substitution, position morphology transformation and adding special characters. Modified samples are transmitted back to the detection model. If a modified sample is predicted as benign, it is labeled as malicious and used for retraining the adversarial and prediction models; otherwise, the sample is sent back to the agent for further deforming.

[6] OWASP ESAPI API: https://owasp.org/www-project-enterprise-security-api/

[7] HTML Purifier: http://htmlpurifier.org

[8] PhishTank: https://phishtank.org

[9] FireHOL: https://firehol.org

The process ends when a number of attempts is reached based on a measured feedback reward formula.

Zhang et al. [141] used an adapted Monte Carlo tree search algorithm (MCTS) for the generation of XSS attacks. Adversarial attacks are generated following a set of bypassing rules enabling hexadecimal encoding, decimal encoding, url encoding, insertion of invalid chars in the middle of tags and case mixture.

Wang et al. [142] proposed a system that starts by applying fuzzing to build a dataset of malicious and benign samples. Those samples are used as inputs to an adversarial attack model based on reinforcement learning. Two agents based on Soft-Q learning algorithm are proposed for the generation of adversarial attack samples. The former applies escaping actions on HTML tags while the latter performs actions on JavaScripts. Those actions include: string substitution, char coding and string addition. The algorithm has demonstrated improvement, but it suffers from coding misuse and a low escaping rate. Chen et al. [143] proposed a framework to mitigate these limitations. They investigated the capability of deep learners and existing tools in handling adversarial attacks. The proposed framework used the Soft Actor-Critic (SAC) reinforcement learning algorithm for generating adversarial attack vectors. This algorithm incorporates a collection of 27 basic mutation operations applied based on well-established rules, providing a 6% increase of the escaping rate compared with Wang et al. [142] work.

Tariq et al. [47] proposed a mixed approach for the detection of XSS attacks based on genetic algorithm, statistical inference and reinforcement learning. The aim was to improve the detection of new XSS attacks. The proposed approach uses a genetic algorithm to check the distance of each payload to malicious and benign samples. If the algorithm fails due to overlaps, a statistical inference module is used. In addition, a threat intelligence (IP addresses and malicious domains) is used to enforce the accuracy of the prediction. A payload is considered safe only if the three detection models predict it as safe, otherwise the payload is considered malicious. Reinforcement learning is used to add and update the samples to recognize more new payload attacks. The proposed approach reached 99.89% of accuracy.

For benchmarking, Mokbal et al. [144] proposed an algorithm named C-WGAN-GP for oversampling datasets without overfitting effects. The proposed algorithm generates synthetic but valid and reliable samples of the minority class. They are also indistinguishable from real XSS payloads.

5.4.3. *XSS attack defense techniques*

Dynamic defense techniques provide mechanisms to protect systems against realtime attacks, block the execution of malicious actions embedded on payloads and prohibit their propagation. The study of the literature distinguishes three main mechanisms: *filtering, moving target defense* and *sandboxing*.

Filtering

Filtering is a blocking based technique. It is a simple way to defend against cyberattacks including XSS. Through filtering, distinguished XSS attacks from benign scripts are blocked automatically. Several filtering based approaches are found in the literature, they only differ on the way to distinguish XSS attacks.

Whitelist/blacklist based filtering. Through identifying and listing all benign/malicious scripts, XSS attacks can be distinguished and blocked by the system. Actually, this is an infeasible solution since listing all malicious or benign scripts cannot be established in practice. To alleviate this problem, all scripts are considered malicious except those used by developers to construct the target web applications.

Wurzinger et al. [162] proposed a proxy based filtering technique. All scripts used by developers in web applications are encoded and every HTTP response is received and sent to a modified browser in the proxy server. This later checks for any clear script. Those scripts are identified as malicious and blocked. If no clear script is generated, original encoded scripts are decoded and clear HTTP responses are safely delivered to users. The solution introduces a large overhead due to frequent decoding of scripts at runtime and thereby not suitable for high performance web services.

Mitropoulos et al. [163] proposed a client-side filtering approach. The JavaScript engine of browsers is extended with a transparent script interception layer that identifies every script not in the list of valid scripts previously fixed by developers for each page; scripts not on the list are simply blocked. To avoid false positives caused by slight changes and dynamic code generation, the layer uses fingerprints previously created on the server-side when comparing scripts. For robustness, when script elements are altered or new scripts are added on the server side, a new fingerprint generation phase is required. Scripts not matching any fingerprint are blocked and an alert is sent to the administrator. The solution causes a negligible overhead (less than 0.05%) but the initial and effective creation of fingerprints using dynamic analysis is hard. In addition, the proposed solution fails to detect all mimicry attacks (JSM-XSS) and requires the modification of browsers to integrate the script interception layer.

Gupta and Gupta [164] used a database of attack vectors associated to each injection point. These are identified by a hybrid analysis: static analysis is performed through parsing files to identify all the injection points and dynamic analysis is performed through injection of malicious attacks and observing the resulted behavior. At runtime, HTTP requests with injected payloads are analyzed by the server. Any match is reported as an attack and the requests are blocked. The database is updated whenever new attack vectors are detected. The experiment showed that the solution suffers from higher false positive and negative rates.

Chaudhary et al. [165] proposed a proxy based filtering approach that uses a blacklist of XSS attacks. HTTP responses are transformed in a way that vulnerable scripts are isolated and saved in a separate file. They are next decoded and grouped following the Levenshtein distance similarity to reduce the refining phase. Finally, they are matched to XSS attack vectors stored in a repository. Matched scripts are sanitized making use of XSS filtering APIs and safe responses are delivered to users. Unfortunately, the solution depends on an attack repository which prevents the detection of all new attacks even with the frequent update if such repository.

Pazos et al. [166] proposed a client-side solution that makes use of well-identified vulnerabilities recognized by analysts deployed in a form of well-formed signatures. A browser extension receives HTTP responses, matches injection points and apply proper sanitizers matching signatures and injection points from already known vulnerability reports. The system maintains a database of such signatures for reuse. The aim is to reduce time from zero day attack to the deliverance of patches where they are directly applied by the browser. The solution causes a considerable overhead (from 10% to 50%), depends on delivered signatures and does not support complex sanitizers.

Regular expression based filtering. Instead of explicitly set the list of scripts to be allowed or blocked, regular expressions are used to describe general patterns of such scripts. Those patterns are used by client or server-side filters to match and block malicious scripts at runtime.

To defend against CS-XSS, Gebre et al. [167] proposed a server-side filter making use of regular expressions modeling dangerous scripts that can be injected in specific HTML elements. The proposed filter has low overhead (60 ms) and produces no false positive but it is unable to distinguish benign HTML tags in PDF files. Moreover, designed regular expressions do not cover encoded and obfuscated scripts.

Javed and Schwenk [168] proposed a regular expression based filter to defend against XSS attacks on mobile based web applications. They improved the set of regular expressions defined in [82] for vulnerability detection and added new ones. In total, twenty-five patterns are defined. The filter is implemented as a JavaScript function embedded on the code of mobile web applications to filter inputs at runtime. The proposed filter is not suitable for desktop based web applications due to their complex nature and the significant use of AJAX based interactions.

Table 8
XSS attack detection through HTTP traffic monitoring.

Study	Detection scheme	Location	Attacks	Pros	Cons
Johns et al. [145]	For RXSS, they match scripts extracted from HTML requests and responses after removing all encoding and performing a threshold-based similarity match. For SXSS, a model is trained on scripts included in the web-application after the execution under benign data and any detected deviation is reported as an attack.	server-side	RXSS, SXSS	low false positive rates (0.5% for RXSS and 0.7% for SXSS)	• SXSS detection relies upon a trained model, since it is impossible to cover all the input scenarios, the solution may has higher false positives for more complex web applications. • Relies on a strong assumption that the attacker cannot add external scripts to trusted domains.
Sun et al. [146]	Extracted scripts from HTTP requests are analyzed, recursively decoded and stored. DOM trees of HTTP responses are generated and scripts are extracted from sensitive locations. A similarity is checked between decoded scripts resulting from HTTP requests and responses. A match is reported as an XSS attack.	client-side	WXSS	effective for the detection of self-replicating XSS worms on the client side with reasonably low performance overhead (0.78%–6.75%).	• not suitable for server-side self-replicating worms. • ineffective with the presence of customized obfuscation techniques.
Sundareswaran and Squicciarini [147]	Comparison of generated and stored control flow graph signatures of web pages at the proxy server. Features are extracted from both versions and a threshold value is used to check for potential similarities. A deviation is reported as an attack.	client and proxy	RXSS, SXSS, DXSS	low performance overhead (less than 1%).	• quite higher false positive rate (3.75%). • relies on the presence of clean pages in the server.
Das et al. [148]	Matching runtime execution-sequences to stored legitimate sequences at the training phase. Any deviation is reported as an attack. The legitimate execution sequences list is updated by the administrator when unsuccessful communications are reported by clients.	client-side	RXSS, SXSS, DXSS	satisfactory results obtained for different XSS attacks.	• the automatic update of execution sequences lists is a challenging task. • requires further experiments with real-wold applications.
Yamazaki et al. [64]	Matching generated and stored templates identified by a restoration algorithm. Any match is reported as an attack.	proxy-side	RXSS, SXSS	overhead varies from moderate to low.	• higher false positive rate (20.6%). • relies on the presence of clean pages at the server.

***Rule based filtering*.** Rules can be used to match suspicious links and detect anomalies in input data.

Shanmugam and Ponnavaikko [169] proposed a solution based on adding requirements for data inputs (type, size, allow special char, allow tags). Those requirements are transformed by a tool to XML schema and stored in a database. Therefore, when an HTTP request is received, the input values are extracted and converted into XML objects and then mapped to the correspondent stored schema. If there is a match, the HTTP request is forwarded to the application otherwise an attack is detected and blocked and the user is redirected to an error page. The solution is platform and language independent but requires tough modifications of the server for their integration.

Again, Shanmugam and Ponnavaikko [170] proposed another solution based on adding security attributes to each web application stored in the server. Those attributes identify the security level required for the application such as the maximum number of input chars, the encoding mechanism, and character-set. Those attributes are used to detect any anomaly behavior. The proposed system starts when receiving an HTTP request, the application attributes are loaded and the input is sanitized accordingly. If the input length exceeds the specified maximum number, the input is rejected; otherwise, if the input does not contain any specific char, the call is forwarded to the application, otherwise the input is parsed by separating it into tokens, the resulted vector is verified by checking a list of vulnerability identification rules that makes use of whitelist tags and their attribute values, if one of the rules is not satisfied, the user is redirected to an error page. The time to handle a single vulnerable input is low (0.04 ms). However, the solution does not support all encoding patterns and for robustness, the list of whitelist tags should frequently be updated to follow the development of web programming technologies.

Kirda et al. [171] proposed an integration of a personal firewall that enables users to add new rules, update or remove existing rules. Rules are used to match risky links. Local links are ignored but cross-domain links are detected and users are notified if a suspicious link does not match any rule. Rules are removed through a garbage collector after being un-matched for a long period of time. Moreover, a limited number of external links is permitted otherwise no link is allowed, this is adopted to prevent binary-encoding attacks. The major flaw of the proposed solution is the dependence on users to update rules.

***Content-security policy (CSP)*.** CSP is another type of filtering capabilities. It is a computer security standard that is actually fully or partially supported by most modern browsers. It enables the restriction of resources that the browser can load and execute. A CSP is specified at the server-side, transmitted as HTTP header in HTTP responses and executed at the client-side by browsers. CSPs are commonly used to defend against XSS attacks but are actually used to defend several code injection attacks.

Jim et al. [172] proposed a basic content-security policy named BEEP. BEEP scheme enables the specification of a whitelist of hashed legitimate scripts identified by web page developers. Therefore, all scripts included in HTTP responses are detected prior rendering, hashed and matched to the whitelist of scripts. As a result, only matched scripts are executed. In addition, the policy enables DOM sandboxing making use of *noexecute* nodes to prevent the execution of scripts potentially injected to `div` and `span` tags. The proposed solution is simple to implement and generates no false positives. However, the current implementation requires the manual identification of whitelist scripts by developers and causes a quite large overhead (14.4% for whitelist policies and 6.6% for DOM sandboxing). Johns [173] conducted an

Table 9
XSS attacks detection through machine learning.

Study	Type	Features (#)	Targeted attacks	Best perf.
Single learning				
Nunan et al. [149]	SVM	URL and HTML (6)	RXSS, SXSS, DXSS	ACC = 99.89%
Goswami et al. [150]	kmeans	JavaScript (16)	RXSS	ACC = 98.89%
Mokbal et al. [151]	MLP	URL and HTML (41)	Unspecified	ACC = 99.32%
Mereani and Howe [152]	KNN	scripts from HTTP requests (16)	Unspecified	ACC = 99.86%
Mokbal et al. [153]	SVM	vectorized payloads (96)	Unspecified	F1 = 99.59%
Ensemble learning				
Wang et al. [134]	AdaBoost	URL, HTML and OSN features (12)	WXSS	F1 = 93.90%
Rathore et al. [135]	RF	URL, HTML and OSN features (25)	WXSS	F1 = 97.40%
Mereani and Howe [136]	Stack(SVM-L, SVM-P, RF, NN)	scripts from HTTP requests (62)	SXSS	ACC = 99.97%
Zhou and Wang [139]	Vote(xBN) + Threat intelligence rules	scripts from HTTP requests (30)	Unspecified	ACC = 98.54%
Zhang et al. [137]	Stack(2GMM)	vectorized HTTP requests/responses (200)	RXSS, SXSS	ACC = 96.49%
Nagarjun and Ahamad [154]	HGBC	vectorized payloads (128)	RXXS, SXSS, DXSS	ACC = 99.89%
Li et al. [138]	Semi-supervised RF + weighted-KNN	URL (12)	Unspecified	ACC = 97.30%
Malviya et al. [155]	RF	scripts and HTML (44)	Unspecified	ACC = 100%
Mokbal et al. [156]	XGBoost	URL, scripts and HTML (160 reduced to 30)	Unspecified	F1 = 99.58%
Deep learning				
Kadhim and Gaata [157]	CNN with LSTM layer	vectorized payloads (3000)	Unspecified	F1 = 99.30%
Fang et al. [158]	Bi-RNN with attention mechanism	vectorized payloads (200)	SXSS	F1 = 99.00%
Chaudhary et al. [159]	CNN	Tagged payloads + PCA (100)	RXSS, SXSS	F1 = 99.37%
Liu et al. [160]	GCN = Graph Convolutional Network	vectorized payloads (200)	RXSS, SXSS	F1 = 99.70%
Pan et al. [161]	GCN = Graph Convolutional Network	vectorized payloads (300)	RXSS, SXSS	F1 = 99.86%

empirical study showing the limitation of CSP 1.1 in defending against JSM-XSS attacks and compromising whitelisted domains. To overcome such loopholes on the use of CSP, the author proposed a system named PreparedJS. The proposal is based on templating scripts and cryptographic script checksums. By templating scripts, placeholders where required data needed to be injected are marked with a specific syntax. Each placeholder is stored together with a list of allowed values in a(JavaScript Object Notation (JSON) format. By cryptographic script checksums, prepared templates are hashed, so that every script received at the browser is hashed and matched to allowed scripts. The list of allowed script checksums are included in the security policy. The proposal has low overhead (from 54 ms to 148.6 ms) but requires considerable modifications of web applications.

Stamm et al. [174] introduced another basic CSP to protect web applications against XSS attack. The proposed scheme enables blocking inline scripts and prevents the transformation of strings into codes by blocking calls to the *eval()* function. The proposed solution requires the modification of web applications to export inline scripts to external files. To alleviate this issue, Doupé et al. [175] proposed a similar but automatic system named DeDacota. The aim was to separate code from data of web application pages and enforce the browser, through a CSP, to block any execution of inline scripts. However, no formal proof of correctness is provided for the tool.

Fazzini et al. [176] proposed a system that automatically generates CSPs for web applications. Starting by a dynamic training with the help of input tests and taint-analysis, the system learns trusted and untrusted parts of web pages making use of annotated DOM trees. The system infers a policy that blocks untrusted parts while the source code of the application is transformed to meet the inferred policies. The transformations include moving inline scripts into external files and prevent inline scripts in the CSP. The proposed solution causes no significant overhead (less than 59 ms) but it may generate false positives that require manual intervention of developers. In addition, the solution cannot deal with source code statements that are dynamically generated by client-side codes.

Pan et al. [177] also proposed a system for the automatic generation of CSP policies for web applications. The proposed system is firstly trained for the inference of script templates. Secondly, web pages are rewritten so that benign and matched scripts of inferred templates are extracted and stored in a trusted subdomain and included as external scripts. This way only external scripts from the trusted subdomains are allowed. Dynamically generated scripts are matched to the templates before allowing their execution. If matched, the web page is modified on the fly to call the correspondent external file including the originally inlined script. This solution causes a moderate overhead (9.1%) but suffers from the cold start problem.

To protect against DXSS, Iqbal et al. [178] proposed a CSP that prevents unauthorized alteration of DOM objects generated at the browser. The CSP policy is enforced by a DOM monitoring module installed in secure browsers. This module monitors client-side DOM behaviors and authorizes or blocks incoming requests. This enables a server-side control on the modifications performed on the DOM at the client-side. To prevent obfuscated requests, the DOM monitoring module deobfuscates HTTP requests before reaching the DOM-API. The proposed solution is resilient to obfuscated malicious requests and causes negligible overhead. However, it requires modifications of browsers and manual specification of policies.

Xu et al. [48] proposed a new CSP named JavaScript CSP (JSCSP) that uses refined rules to defend injections to specific tags or elements instead of whole pages. It also enables the automatic generation of rules. When the JSCSP is deployed, DOMs of received pages are blocked temporarily and safer DOMs are generated, the policy rules are checked and malicious scripts are removed, then the original DOM rendering is enabled. The evaluation showed that JSCSP is compatible with different browsers but depends on the availability of clean pages, otherwise already injected vectors cannot be detected. Moreover, JSCSP is compatible with static websites since dynamic ones allow the addition of new elements dynamically to the DOM. In addition, JSCSP generates high false positives.

Mui and Frankl [179] proposed a quite different solution. They used two different encoding for web applications and user inputs. Trusted web application code is encoded with standard characters where user inputs are encoded using a complementary character encoding. This way, user inputs are easily tracked. Token-based security policies are used to prevent sensitive sinks being reached with special user tokens. This solution replaces the need for introducing sanitizers and causes low overhead (0.17%–1.74%). However, it requires the modification of the server and browser components. It also requires the modification of web applications to remove sanitizers that may cause conflicts with the proposed scheme.

Runtime taint-tracking. Runtime taint-tracking enables controlling whether attacker injected data in URLs reach sensitive sinks and cause the browser to execute their incorporated scripts.

Vogt et al. [180] used runtime taint-tracking to prevent user sensitive data from being sent to third-parties. The JavaScript engine of the browser is modified to support dynamic (bytecode) taint propagation. Therefore, whenever a JavaScript program attempts to transmit sensitive data, the system checks whether the host of the loaded page and the host to which sensitive data is sent are from the same domain. If it is not the case, users are alerted to take the appropriate actions. This is only suitable for experienced users.

Stock et al. [181] used runtime-taint tracking to stop parsing scripts other than those used by trusted developers. For this sake, the JavaScript engine is modified to enable the tokenization of scripts, the identification of script origins and the ignorance of any code containing non-literals. The rendering engine accepts only tainted data to appear in the protocol and the domain of remote URLs from the same origin. In addition, an API is provided to specify when the application should activate the dynamic generation of elements to avoid blocking desirable behaviors. The solution causes a moderate overhead (between 7% and 17%) but requires extensive modifications of browsers.

Upload filtering. Upload filtering is a server-side technique that checks file contents for potential injection of malicious scripts when they are uploaded in the server. Suspicious detected files are either modified by removing injected scripts or simply rejected. It is a powerful technique to defend against CS-XSS. It is explored by two studies. Barth et al. [182] implemented an upload filter constituted of 34 HTML signatures. So that, any file content matching any of those signatures is reported as a suspicious file and blocked by the server. Only the initial bytes of files are inspected while scripts can be inserted anywhere. To overcome this limitation, Barua et al. [183] developed a two stage detection filter. Firstly, the MIME type is detected from file extensions and metadata. If the MIME type cannot be detected or it is not whitelisted, the upload is rejected. Secondly, files with known and benign MIME types are subjected to a content analysis. This later incorporates an encoding process and a content parsing for locating JavaScript method calls. The presence of JavaScript tags indicates the suspiciousness of files, thereby a browser emulator is invoked to determine any malicious actions when downloading. However, the solution causes large overhead (from 353% to 3030%).

Moving target defense

Moving Target Defense techniques (or MTD for short) are designed to reduce the success rate of attacks by continuously shifting the configuration and parameter values of running applications. This hardens the understanding of structure and behavior of web applications which increases the uncertainty of adversaries and protects against traditional attacks. Randomness is the key success of MTD techniques since deterministic changes can be learned by attackers and hence permit them to design new successful attack vectors. MTD techniques has shown their ability to defend against sophisticated and complex cyber-attacks [184]. For defending XSS attacks, eight studies are identified by this review:

Nadji et al. [185] proposed a client–server solution that prevents altering the structure of trusted contents by untrusted data. At the server-side, trusted and user-generated data are separated by annotating trusted data using randomly generated markup primitives. At the browser-side, untrusted user-generated data can easily be distinguished at the parsing level, and then isolated and tracked dynamically. Server-side policies are used to confine untrusted data. This solution has low false positives, browser-level and server-level overheads (from 1.1% to 1.85% at the browser and from 1.2% to 3.1% at the server). It requires modifications of web applications, servers and browsers to support the parser level isolation and interpretation policy enforcement.

Athanasopoulos et al. [186] proposed a system that automatically extract all JavaScripts of web pages, encrypt them and transpose them to a randomly generated domain. The decryption key is transmitted to the client side through action-based policies in the HTTP header. Therefore, attacker injected codes cannot bypass the policy. The evaluation process showed that the system has a negligible computational overhead but the integration of the system requires modifications on browsers.

Shariar and Zulkernine [187] proposed a similar approach to deterministic moving target defense techniques. Each stored web application file is inspected where injection points are firstly identified and comments are inserted before and after each point. Injected comments are stored with the expected features of the enclosing points (i.e., number of tags, number of attributes). For every received request, the server generates the response instrumented with injected boundaries, and the features are calculated and compared with the stored ones, each deviation indicates a potential XSS attack. In such case, malicious contents are removed. Injected boundaries are removed before the response is sent to the browser. The solution produces false positives that reach up to 5.2% with a reasonable response-time delay that ranges from 2% to 6%. However, it is language dependent since appropriate parsers are required to identify every injected point. In addition, the deterministic nature of injected comments may cause attackers to learn the process and launch adversarial attacks. A similar but improved approach has been proposed later on by Gupta and Gupta [78]; in this latter work, injected comments are formed from random generated strings with runtime sanitizer instrumentation for avoiding the execution of malicious scripts. However, the new proposal produces higher false positives that range from 10% to 15%. Later on, a cloud based version for OSN web applications is proposed by the same authors in [188,189].

Van-Gundy and Chen [190] proposed an MTD based solution that enables browsers to distinguish between benign and malicious contents of web pages. The server used a function to randomly generate strings used as prefixes to the (X)HTML tags in response of each HTTP request. This way, every injected content will be easily distinguished by the browser. Those contents are blocked making use of a security policy transferred by the server and checked by the browser before rendering the page. The solution proposal is simple and produces no false positives with moderate overhead (14%–20%). But, it requires server-side modifications and manual specifications of policies for each application.

Niakanlahiji and Jafarian [191] proposed a different MTD technique. They modify web applications by adding a new attribute named *runtimeID* to suspicious tags with randomly generated hex value. The head of each page is also modified to include a script that will be executed for every user request. The role of that script is to generate a random hex identifier and add it to each *runtimeID* attribute in the page. At the browser side, new added elements without a *runtimeID* attribute

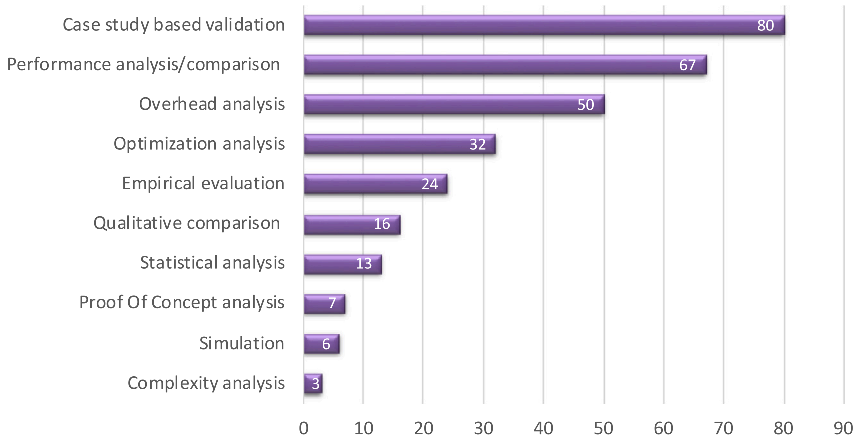

Fig. 14. Distribution of validation techniques.

or with different *runtimeID* value are considered as malicious and are removed automatically. The solution produces no false positives and has low average overhead (4 ms). Unfortunately, it requires manual interventions to add the *runtimeID* attribute to sensitive elements.

Sandboxing/isolation

Sandboxing is an isolation technique that enables the full control of running applications and prevents suspicious actions to be propagated. Cao et al. [192] proposed a refined sandboxing technique to prevent WXSS attacks. Rendered pages are isolated at the client-side. Every web page is divided into different and separate parts named views according to a specific strategy. Each view has a list of predefined authorized actions. Therefore, views cannot perform and request actions from other views without authorization. For isolation, pseudo-domains are associated to views and each view has a list of controlled capabilities. This way, every required connection from one view to another is controlled by an authentication mechanism. For example, a request to the server to modify a content-profile will be verified and blocked if it is originated from an infected page (not authorized action). The request emanating from a view that has no capability to send it, is blocked. This can be implemented by techniques like per-url session tokens and referrer-based view validation. The technique has a reasonable latency and memory overhead (about 1% and 4% respectively) but involves modifications of web applications to define views and their correspondent authorized actions.

XSS attack detection and defense techniques

Nine studies included in this review proposed a runtime detection with defense techniques. Remarkably, runtime sanitizer instrumentation is the dominant approach adopted as a defense technique. Table 10 summarizes those studies highlighting their targeted attacks and limitations.

5.5. What type of evidences are used to validate solution proposals? (RQ5)

The diversity of solution proposals enabled the use of different validation techniques and data sources. In this section, we describe the different validation techniques used to check the effectiveness of proposed solutions as well as sources used for experimentation. This allowed us to evaluate the quality of studies included in this review and may help researchers to replicate existing studies.

5.5.1. Validation techniques

The adopted data extraction process allows the identification of variant validation techniques adopted by existing studies. Case study, quantitative comparison with similar works and performance overhead analysis are the most adopted techniques as showed in Fig. 14.

Fig. 15 shows refined results where adopted validation techniques are distributed per type of contribution to tackle the attack. Despite the importance of overhead analysis in runtime detection of attacks, several studies focused only on their detection performances omitting such important factor. This prevents fair comparison of accurate validation of existing proposals and checks their usefulness in practice.

5.5.2. Data sources used in validation

Different publicly available sources for XSS information including recent attacks and attack vectors have been used. Those sources are given in Table 11 together with the list of studies using them. Note that only active links are reported in the table, dead links are ignored. It can be depicted that XSSeed, OWASP and HTML5 Security CheatSheet are the most used sources for retrieving malicious XSS payloads. Alexa top sites are the most experimented URLs and sources to extract benign scripts. Only four complete datasets are made available in Github, Kaggle and figshare.

6. Issues and open challenges

In this section, we discuss limitations, issues and open challenges preventing the proper handling of XSS attacks. Those issues and challenges are derived from the findings of the present review.

6.1. Lack of appropriate vulnerability repair mechanisms

Handling XSS vulnerabilities is a first step toward defending against XSS attacks. But vulnerability repair has earned less attention compared with detection and mitigation. Fig. 8 together with Fig. 13 indicate that just two studies (1.27%) focused on vulnerability repair, whereas forty-nine studies (31.21%) were devoted to vulnerability detection and six studies (3.82%) were devoted to vulnerability mitigation. Mitigation techniques are important but have several unstudied flaws. Secure APIs require much time for their development, feasibility check, standardization and adoption by different browsers. Secure templating are framework dependent and they are still in their infancy stage (see Section 5.4.1 for more details about the limitations of proposed techniques). Sanitization is an intuitive solution to eliminate XSS vulnerabilities but more automatic tools and guidelines are need to be provided for their appropriate and easy usage by inexperienced developers. Therefore, automatic and easy to use tools for detection with repair capabilities of XSS vulnerabilities are still an open research direction that has not yet been sufficiently explored.

Table 10
XSS attack detection and defense techniques.

Study	Detection method	Defense method	Attacks	Limitations
Bisht and Venkatakrishnan [193]	HTTP traffic monitoring and matching shadow and real generated pages. Shadow pages are obtained through transmitting HTTP requests with benign string inputs.	Remove all scripts not present in intended pages.	Unspecified	• performance overheads ranged from 5% to 24%. • produces false positives when attacks use non-Firefox quirks.
Bates et al. [194]	HTTP traffic monitoring and matching scripts from HTTP requests and responses after recursive decoding	Refuse to deliver matched scripts to the JS engine.	RXSS	• limited to the detection of single injections omitting multiple injections. • requires modifications on the browser.
Pelizzi and Sekar [195]	HTTP traffic monitoring and matching scripts from HTTP request and response parameters	Refuse to deliver matched scripts to the JS engine.	RXSS	• requires the modification of the browser to check the execution permission of each script. • unable to detect scripts subjected to extensive string transformations by web applications.
Gupta and Gupta [9]	HTTP traffic monitoring and matching scripts from HTTP requests and DOM trees generated for HTTP responses	Runtime context-aware sanitizer instrumentation.	WXSS	• low detection rate (F1-score less than 95%).
Gupta and Gupta [196]	HTTP traffic monitoring and matching scripts from HTTP requests and responses	Runtime context-aware sanitizer instrumentation.	RXSS, SXSS, DXSS	• no performance overhead analysis is performed.
Gupta et al. [10]	HTTP traffic monitoring and matching scripts stored and generated DOM trees	Runtime context-aware sanitizer instrumentation.	DXSS	• quite large overhead due to runtime nested auto-context-sensitive sanitization.
Lalia and Sarah [197]	HTTP traffic monitoring and matching script features extracted form generated HTTP responses with those stored after performing a static analysis.	Runtime context-aware sanitizer instrumentation.	RXSS, SXSS	• not suitable for encoded scripts and CSS file scripts. • relies on the presence of clean pages at the server. • high false positive and negative rates due to the ignorance of dynamically generated scripts.
Gupta and Gupta [11]	HTTP traffic monitoring and matching stored sanitized JS variables with unsantized ones generated from HTTP responses. Stored sanitized JS variables are collected through performing a dynamic analysis.	Runtime context-aware sanitizer instrumentation.	WXSS	• low detection rate (less than 97%).
Chaudhary et al. [49]	HTTP traffic monitoring through matching stored and generated scripts.	Action authentication with Runtime context-aware sanitization.	WXSS	• lack of performance overhead analysis.
Krishnan et al. [198]	Stacking ensemble learner that integrates three traditional and three deep learning classifiers for the detection of malicious URLs.	Basic filtering operations: URL encoding, removal of suspicious words, and elimination of special characters embedded within URLs.	RXSS, SXSS, DXSS	• lack of performance overhead analysis. • not tested on real-world scenarios.
Chaudhary et al. [6]	By runtime monitoring of HTTP traffic and parsing HTTP requests and responses, strings form HTTP queries and well-identified locations in parsed HTTP response are extracted converted into clear forms, which are then compared to a blacklist of attack vectors.	Identified contexts of malicious strings are used to invoke appropriate filtering API routines.	RXSS, SXSS	• inherits the limitations of blacklist and filtering approaches.

6.2. Limitations of vulnerability detection techniques

For the detection of XSS vulnerabilities, both static and dynamic analysis are explored by existing studies. According to Fig. 13, twelve static analysis approaches and sixteen dynamic analysis strategies were suggested. Those approaches have several known limitations. Static analysis based approaches require the availability of the code of tested web applications and suffers from reporting high false positive rates [199]. Besides those general limitations, individual static based solutions have other specific flaws. Model driven-based code auditing is unsuitable for large web applications; manual specification of models becomes complicated; besides, load and manipulation of large models is cost-effective. Machine learning based code auditing suffers from the lack of interpretability of results; the black-box nature of classifiers prevents understanding their classification processes. The use of gray-box classifiers such as J48 used in [75] may alleviate this issue since paths can be extracted from trained models which enables the understandability of the classification scheme. Static taint-analysis suffers from the inability to handle dynamically generated scripts at browsers which is a common practice that is widely used in advanced web applications. In the other side, dynamic analysis has also some limitations; as the number of injection points increases, it takes longer to detect all XSS vulnerabilities. Moreover, it produces high false negatives due to their reliance on a set of payloads that are often unable to cover all possible paths. Despite the extensive interest in test-case generation (eighteen studies are reported in this review, see Fig. 13 and Section 5.4.1 for more details), those techniques are still ineffective for the detection of all real-world scenarios specially with the use of obfuscations. Hybrid analysis provides a trade-off between static and dynamic analysis, however only three studies (see Fig. 13) are found in the literature, thereby we advocate more studies in this direction. Moreover, much interest should be given to handle vulnerability flaws across multiple pages; only few studies are addressing such an issue.

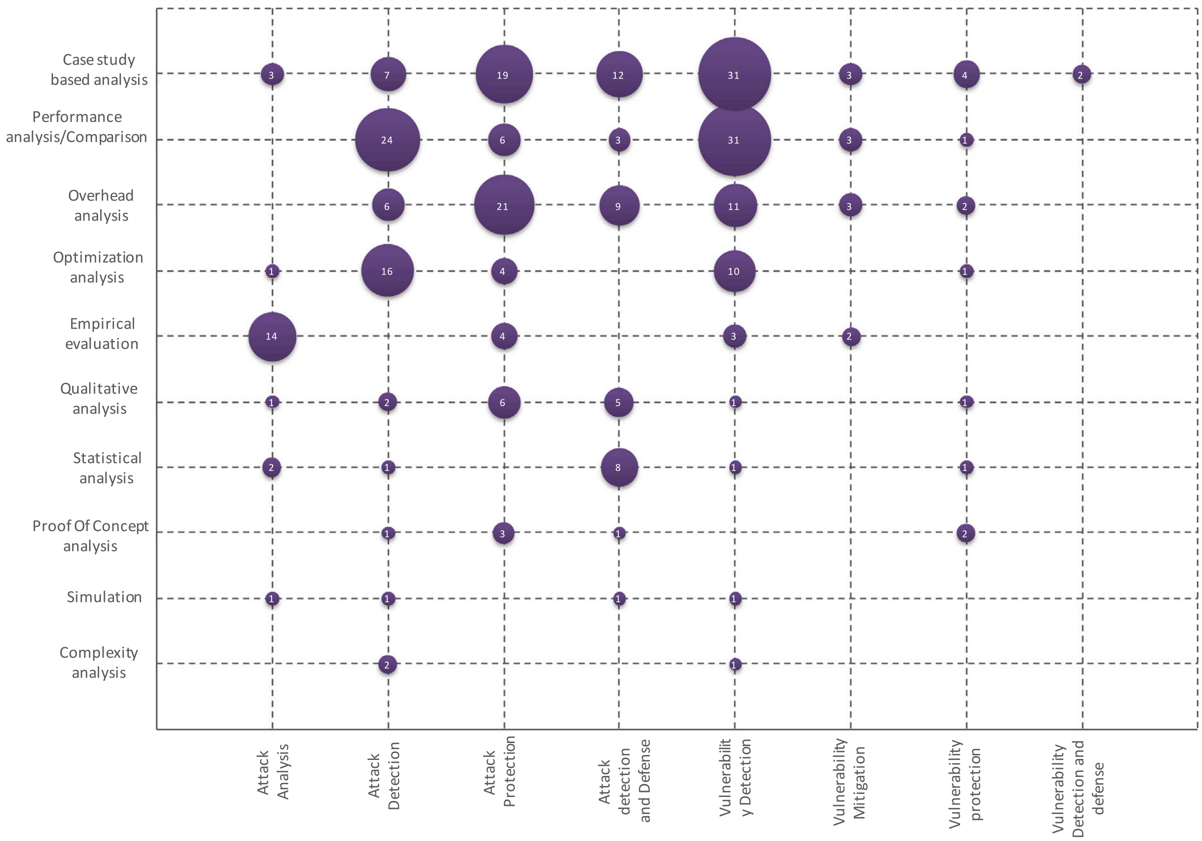

Fig. 15. Distribution of validation techniques per contribution to tackle XSS attacks and vulnerabilities.

6.3. Bias of interest toward basic XSS attacks

Several XSS attack types are reported and explored in the literature (see Section 5.3). However, the different types are not treated equally (see Fig. 10). The study revealed a bias toward the three basic attacks (i.e., RXSS, SXSS and DXSS). The other attacks are not sufficiently studied and more attacks may be in the way. This can be justified by the fact that basic attacks are well-recognized by web security communities and are well-documented. The experience showed that no single solution is able to handle all the attack types, hence more appropriate solutions should be explored to defend against other variants of XSS attacks.

6.4. Limitations of attack detection techniques

The study revealed that machine learning is the widely explored approach for the detection of XSS attacks (see Fig. 8 and Section 5.4.2). We agree that machine learning is a promising approach for the detection of cybersecurity attacks including XSS. However, three main obstacles hinder their usage in practice: *lack of interpretability of their predictions*, their *susceptibility to adversarial attacks* and *lack of benchmarks* [200,201]. The effective adoption of machine learning for cybersecurity attack detection requires to properly mitigate these three problems. In the review, only two studies have been found addressing the interpretability of their proposed models [139,152]. In [139], Bayes Networks are adopted since they provide clear semantics that enable learning probability distributions from data [202]. In [152], the authors proposed deriving explainable rules from black-box models that make the predictions generative and explainable. The latter approach is limited to models using binary features which does not fit to all discriminating features explored in the literature. Regarding susceptibility to adversarial attacks, only five studies addressed the issue, reinforcement learning is the most adopted solution to improve the performance of machine learning models through learning and generation of adversarial attack vectors and retrain models on those attacks (see Section 5.4.2). More techniques should be explored in this direction. Additionally, benchmarking is a long standing problem for machine learning based proposals. Benchmarks enable fair comparison of existing solutions and the development of more robust models [201]. In [144], the authors cooperate by proposing a new oversampling algorithm for XSS datasets. More researches are also advocated to deal with such problem.

6.5. Limitations of attack defense techniques

Due to the rapid development of attack tactics, traditional filtering techniques such as white/blacklist, Regex pattern matching become inefficient for the detection of all XSS attacks. They need to be frequently updated whenever new attack vectors are reported. Surprisingly, despite the prevalence of web applications with file upload capabilities, such as online social networks, upload filtering is getting less attention. Only two studies covered the issue but with unsupported limitations [182,183] (see Fig. 13 and Section 5.4.3). We advocate more studies that explore properly the upload filtering technique. In the other side, content security policies (CSP) is a powerful and robust filtering solution, if used properly, to defend against XSS attacks. The ideal solution requires to deal with scalability, compatibility, dynamic generation of elements and automatic generation of policies. Those issues are partially addressed by contemporary studies (see Section 5.4.3). The study of Xu et al. [48] is the only contribution targeting the scalability issue of CSP policies where fine-grained rules are enabled by the proposed solution. Fine-grained policies enable the specification of allowed and/or blocked scripts at elementary and specific tags. The automatic generation of CSP policies for web applications is also mandatory for two reasons: (1) avoid potential errors caused by manual

Table 11
XSS attack vectors data sources: M = Malicious, B = Benign, MB: Mixed, L: Live link, D: Dead link.

Source	Link	Type	Status	Content	#studies
XSSed project	http://www.xssed.com/	M	L	XSS payloads	19
Rsnake XSS Cheat Sheet	http://ha.ckers.org/xss.html	M	D	XSS payloads	25
Technomancie.net	http://xss2.technomancie.net/vectors/	M	D	XSS payloads	8
@XSS Vector Twitter Account	https://twitter.com/XSSVector	M	D	XSS payloads	9
Open Bug Bounty	https://www.openbugbounty.org/	M	L	Vulnerable URLs	3
DMoz Archive	http://dmoztools.net/	B	L	Non-vulnerable URLs	2
ClueWeb+	https://www.lemurproject.org/	B	L	Non-vulnerable URLs	3
Alexa	https://www.alexa.com/topsites	B	L	Non-vulnerable URLs (stopped since May 2022)	14
Github	https://github.com/duoergun0729/1book/tree/master/data	M	L	XSS payloads	3
	https://github.com/IramTariq/XSS-attack-detection	MB	L	Vectorized XSS and benign scripts	1
	https://github.com/ajinabraham/OWASP-Xenotix-XSS-Exploit-Framework	M	L	XSS payloads	1
	https://github.com/bsmali4/XSSfork	M	L	XSS payloads	1
	https://github.com/fuzzdb-project/fuzzdb	M	L	XSS payloads	2
	https://codeload.github.com/foospidy/payloads/zip/master	M	L	XSS payloads	1
	https://github.com/stivalet/PHP-Vulnerability-test-suite	MB	L	Vulnerable and safe PHP synthetic test cases	1
	https://github.com/payloadbox/xss-payload-list	M	L	XSS payloads	3
	https://github.com/pgaijin66/xss-payloads/blob/master/payload/payload.txt	M	L	XSS payloads	1
	https://github.com/WhiteRabbitc/Wooyun-Email-XSS-Dataset/tree/master/malious-sample	M	L	XSS payloads	1
	https://github.com/ismailtasdelen/xss-payload-list	M	L	XSS payloads	1
	https://github.com/danielmiessler/seclists/tree/master/miscellaneous/web	M	L	XSS payloads	1
	https://github.com/yahoo/webseclab	M	L	1	
	https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/XSS/XSS-RSNAKE.txt	M	L	XSS payloads	2
	https://gist.github.com/kurobeats/9a613c9ab68914312cbb415134795b45	M	1		
Kaggle	https://www.kaggle.com/datasets/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning	MB	L	XSS and benign scripts	2
	https://www.kaggle.com/shawon10/url-classification-dataset-dmoz	MB	L	XSS and benign scripts	1
OWASP project	http://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet	M	L	XSS payloads	18
HTML5 Security Cheatsheet	http://html5~sorg/	M	L	XSS payloads	15
Vulnerability Lab	https://www.vulnerability-lab.com/	M	L	Software products protecting against XSS	9
CWE+	https://cwe.mitre.org/data/index.html	M	L	Known vulnerabilities	1
figshare	https://figshare.com/articles/dataset/XSS_dataset1_csv/13046138/4	MB	L	Victorized XSS and benign URLs	1
Impact Cyber Trust	https://www.impactcybertrust.org/dataset_view?idDataset=940	M	L	Generated Web traffic	1
PastBin	https://pastebin.com/BdGXfm0D	M	L	XSS payloads	1
JS CheatSheet	https://htmlcheatsheet.com/js/	B	L	Benign scripts	1
Chef Secure	https://chefsecure.com/blog/the-12-exploits-of-xss-mas-infographic	M	L	XSS payloads	1
Exploit database	https://www.exploit-db.com/	M	L	XSS payloads	2
Information Technology Laboratory	https://nvd.nist.gov/vuln	M	L	Known vulnerabilities	1

specifications and (2) handle complex and large scale web applications. The automatic generation of CSPs is addressed in [175–177]. However, those solutions have several limitations related to high false positive rates and performance overheads. Therefore, more improvements are required to reach the ideal CSP policy based solution. Moreover, dealing with dynamic element integration is mainly treated by blacklisting the *eval()* function such in [48]. The same problem is faced by MTD based techniques. More solutions should be explored such as adopting effective combinations of static and dynamic analysis for the automatic generation of policies. This way, dynamically added elements can be covered by policies as well as randomization mechanisms used by MTD based approaches. There is still an obstacle facing the effectiveness of CSP policies which is their susceptibilities to CR-XSS attacks (see Section 5.4.3). Consequently, more studies are needed to explore how to cope with CR-XSS attacks in CSP based policies.

6.6. *Bias toward JavaScript based XSS*

Cross-site scripting attacks are primarily caused by the execution of injected malicious scripts making benefits of unsanitized input data. Those scripts are not necessary written in JavaScript, other scripting languages such as VBScript, ActiveX, Flash, and even CSS can be used. However, a remarkable bias is shown toward XSS attacks caused by JavaScripts (see Fig. 12). The bias is understandable because of the popularity of JavaScript. However, for completeness and robustness, other XSS vectors written in other scripting languages should also be taken into account. An initiative was taken in [193], by preventing the execution of scripts embedded on Flash objects. Bensalim et al. [120] also considered attacks that can come from embedding scripts into JSON files. More deep investigations are required to deal with all XSS attack vectors. In addition, dealing with HTML5 specific features when designing new defense mechanisms starts getting much attention.

Table 12
Brief answers to the research questions.

Research questions	Answers
RQ1. How has interest in addressing XSS attacks evolved since its unveiling in 1999?	Although XSS attacks are long-standing security threats, they have only garnered increased interest since 2016. Researchers from all over the globe have contributed significantly to the literature on XSS, with a considerable number of studies originating from China, USA, and India. The prevalence of high-quality conferences in the cybersecurity field and their fast publication timeline have led to a substantial number of published papers on conference proceedings. In recent years, there has been a notable shift toward publishing more research in journals with rapid reviewing processes, such as the Computer and Security journal, enabling swift accessibility of research findings to fellow scholars. Top conferences publishing research about XSS are listed in Table 5.
RQ2. What type of research has been published addressing the XSS issue?	Existing studies comprehensively address various perspectives in dealing with XSS attacks, encompassing analysis, experimentation, detection, defense, and prevention, albeit with varying emphasis on each aspect. Notably, a bias toward XSS attack prevention has been observed, with a greater focus on XSS vulnerability detection, while comparatively less attention is directed toward XSS vulnerability repair.
RQ3. What types of XSS attacks are addressed by contemporary studies?	In addition to the common known types of XSS attacks (RXSS, SXSS, and DXSS), the literature reports the existence of seven more variants (MXSS, WXSS, JSM-XSS, CS-XSS, CR-XSS, UXSS, and XAS), each possessing with its own peculiarities. We proposed a comprehensive taxonomy of these attacks in Fig. 9, which is formulated based on a thorough analysis of the nature and origin of each attack. Despite the dangerous consequences associated with these various XSS attack variants, it is noteworthy that a higher level of interest has been directed toward the three basic XSS attacks in the research community.
RQ4. What techniques have been used to tackle the XSS issue?	Various approaches and techniques have been developed to address XSS attacks, tailored to different types of interventions: prevention, detection, and defense. Fig. 13 presents a comprehensive taxonomy of the proposed approaches, categorized by their respective intervention types. Machine learning, with its diverse techniques such as single, ensemble, deep, and reinforcement learning, has been extensively explored for both attack and vulnerability detection. Additionally, filtering approaches, including content-security-policies, have been widely investigated as effective defense mechanisms against XSS attacks. Moreover, numerous traditional and revolutionary test-case generation techniques have been explored for the detection of XSS vulnerabilities in web applications.
RQ5. What types of evidence are used to validate solution proposals?	Several validation techniques have been used depending on the type of solution proposal, with case study-based validation being the most prevalent, often complemented by performance and overhead analysis. However, statistical analysis, which is crucial for rigorous performance comparison and generalization in machine learning models, is comparatively less adopted in existing research. Moreover, data sources for validation primarily include XSSed, GitHub, and OWASP projects, but the absence of benchmark datasets hinders fair comparisons between existing approaches as studies rely on individually collected data from those sources.

Fifteen included studies proposed defense mechanisms that also work with HTML5 embedded attacks [43,120,164,189]. Developers of new defense attacks should follow the same path.

7. Discussion

For over two decades, XSS attacks have remained a persistent and enduring cybersecurity concern. Despite concerted efforts to bolster the security of users, servers, and web applications, these attacks persist as significant threats. In light of this, our study endeavors to make a meaningful contribution to the existing body of knowledge by conducting a comprehensive review and mapping of the state-of-the-art regarding XSS attacks since their inception. To achieve this aim, we meticulously analyzed relevant studies in the literature, addressing five key research questions. Table 12 provides a concise overview of the main findings from the present review, offering brief answers to the five primary research questions.

The present study discusses the different XSS attacks and the approaches taken to defend against them. Existing endeavors have addressed XSS attacks from various perspectives, including prevention, detection, and protection, using diverse techniques and methodologies. However, despite these efforts, XSS attacks continue to evolve, necessitating constant vigilance and innovative countermeasures. The present study serves as a comprehensive guideline for researchers and practitioners seeking to contribute to the mitigation of XSS attacks. By understanding the historical context and the current state-of-the-art, future research can build upon existing knowledge and develop novel solutions to combat this persistent threat efficiently.

The present review's findings unequivocally highlight the multifaceted nature of XSS attacks, presenting a unique and comprehensive taxonomy of XSS attack types and corresponding proposed solutions. In stark contrast to previous reviews that became outdated and focus on specific aspects of XSS attacks (e.g., detection, prevention, mitigation), our investigation encompasses a broader spectrum of XSS attacks. Previously established reviews have reported only four XSS attack types, primarily covering the three basic ones (RXSS, SXSS, DXSS). Notably, the variant MXSS has been reported solely by Chaudhary et al. [22]. Specifically, the present review has expanded this limited scope by shedding light on other lesser-known variants of XSS attacks. Through a meticulous examination of the literature, we have uncovered and discussed additional XSS attack variations that warrant attention and further investigation. By encompassing a broader range of attack types, our review contributes to a more comprehensive understanding of the evolving landscape of XSS threats and enhances the knowledge base for effective countermeasures.

Additionally, while prior review conducted by Hydara et al. [18], identified an overwhelming emphasis on RXSS, and others, like Chaudhary et al. [22] and Gupta et al. [21], highlighted the underrepresentation of DXSS, the updated review shows that all the three basic XSS attacks including DXSS have received extensive attention, but newer variants like CR-XSS and XAS require further investigations. Notably, reviews conducted by Malviya et al. [17] and Rodriguez et al. [15] advocate the adoption of machine learning techniques for detecting XSS attacks, an approach that is found garnered significant attention in our review. On the contrary, the persistent neglect of XSS vulnerability repair techniques, as reported by Hydara et al. [18], remains a critical concern, and we strongly advocate for increased research efforts in this direction.

8. Threats to validity

This section discusses the validity threats of our conducted study. As suggested by Ampatzoglou et al. [203], we discuss three types of threats to validity (study selection, data, research) and we show how we struggled to mitigate them.

8.1. Study selection validity

The identification of relevant papers is the key point of any secondary study. To avoid missing relevant papers, we queried six well-known academic libraries in the field. In addition to those suggested by Kuhrmann et al. [24], we also searched papers in Scopus. The automatic search process is based on the presence of search keywords in titles only. When we applied the search string to full metadata including abstracts, the automatic search produced an unmanageable number of studies; after investigation, we found that most of retrieved papers are irrelevant since they either use XSS to indicate different designations (e.g., low-mass X-ray binary systems) or refer to XSS as a comparable attack or vulnerabilities in their abstracts. Since we focus on papers with a primary focus is XSS, we decided to limit our search process for titles. Since this may affect the results, we completed the search with exhaustive and recursive snowballing considering both titles and abstracts. At the end of the search process, a separate automatic search was made with Google Scholar and no additional relevant papers were retrieved. After relevance and inclusion/exclusion criteria check, we conducted a strict quality assessment process. This was the cause of the elimination of a great number of studies. Specifically, 384 relevant papers were excluded from the review due to their lower quality assessment score, which was below 4. To avoid any bias in the selection of relevant papers and enable replication of this phase, we strived to make the inclusion/exclusion criteria (Table 2) and quality assessment criteria (Table 3) as objective as possible. For example, we did not penalize new publications for their number of citations and we used well-recognized sources for ranking. In addition, each of the other criterion was adopted for a specific purpose as discussed in Section 3.3.2. The majority of the provided criteria fall within this objective category. However, QA2 and QA3 have the potential for subjective judgment. To address this issue, we adopted a two-step screening process. Two different authors independently screened the papers, and in cases where conflicts arose, a discussion session was conducted to reach a resolution. It is worth noting that only five papers were subject for conflict resolution. To ensure a rigorous screening phase, the first two authors, who have previously conducted similar systematic reviews in the field, were responsible for this stage. Additionally, the first author serves as a reviewer for highly reputable scientific journals in the field. His expertise contributes to maintaining a high level of objectivity throughout the selection process. It is worthnoting that the study selection process has two main limitations. Firstly, non-academic studies were not considered due to the lack of well-established sources and the need for distinct quality assessment criteria for grey literature like technical reports and feature papers. Secondly, the review was limited to papers published up until December 2022, as the study began in September 2021, and conducting systematic reviews requires considerable time for analysis and evaluation. Consequently, papers published after 2022 were not included in this review.

8.2. Study data validity

Data extraction from included studies is performed separately by the two first authors. This enabled the identification of potential inconsistencies. When they appear, the third author is called for validation. This may introduce a bias toward the selection of papers. To minimize the effect of this bias, concise templates are used for extraction, this is designed by the authors at the planning phase before conducting the review. In this review, we focused on extracting data specifically aligned with the key research questions to ensure relevance. While we omitted certain details commonly found in previous reviews, such as the applicability of each solution proposal (i.e., client, server, fog, or hybrid), this information is either be explicitly stated or implicitly derived from the detailed descriptions provided for each study. Additionally, we found that sanitization routines are language-specific and continuously evolve with updated language versions. Due to this dynamic nature, the list of sanitization routines are not considered in the data extraction process.

8.3. Study research validity

The results derived from the present study only concerns high quality published studies in academic repositories. Grey literature, non-English and low quality papers are excluded from the review. Specifically several non-English and low-quality papers were found published in the literature, however, sticking to the adopted protocol, those studies were excluded. The exclusion of 384 papers during the quality assessment phase may have some impact on the generalizability of the findings. However, it is worth noting that many of these studies proposed machine learning-based solutions without proper validation or were published in less recognized sources. Despite this, machine learning remains one of the predominant approaches for handling XSS attacks, which mitigates the potential impact of this factor on the overall generalizability of the findings.

9. Conclusion

In this paper, we conducted a systematic mapping and a comprehensive survey studying the advancement in research to tackle XSS attacks. The study is not restricted to a period of time and covered high quality studies published since its discovery. Several studies were found in the literature but a remarkable interest is only observed in the last few years. Despite the diversity of solutions proposed over the years, XSS attacks are still prevalent and targeting new web applications and platforms. The study revealed much attention to XSS vulnerability detection instead of its repair. For securing web navigation, several defense lines should be provided. As a staring point, developers need to be aware of the consequences resulted from ignoring security practices, at the same time, effective tools enabling the automatic detection and repair of XSS vulnerabilities should be made available to facilitate their tasks. Dynamic defense techniques against XSS attacks should also be provided for protecting innocent users when new attacks occur, those techniques are still immature for providing the intended protection level against all types of XSS attacks [204]. Traditional filtering approaches become ineffective regarding the new developed web technologies, more advanced techniques need to be explored. Although the wide adoption of machine learning based techniques for the detection of XSS attacks, existing endeavors only focus on performance analysis omitting three important problems related to cyber-security communities (1) interpretability of prediction results, (2) robustness of models against adversarial attacks and (3) suitability for integration in real-world architectures and platforms. The review also denoted a bias toward basic XSS attacks; this needs to be alleviated by advocating more research targeting other XSS attack variants, specifically WXSS and XAS that are targeting online social networks that may affect wider populations in a short period of time. Moreover, regarding the rapid development of web technologies, XSS attacks written in other scripting languages such VBScript and ActiveX or embedded in advanced web languages such as HTML5 need to properly be studied for completeness and robustness.

Declaration of competing interest

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

Data availability

No data was used for the research described in the article.

Appendix

The detailed list of included studies in this review is given in Table 13.

Table 13
List of included studies with their detailed quality assessment scores: J: Journal paper, C: Conference paper.

Ref.	Year	Type	QA1	QA2	QA3	QA4	QA5	Total QA Score
Huang et al. [69]	2004	C	0.5	1	0.5	1	1	4
Jovanovic et al. [79]	2006	C	1	1	0.5	1	1	4.5
Shanmugam and Ponnavaikko [169]	2007	C	1	1	0.5	0.5	1	4
Shanmugam and Ponnavaikko [170]	2007	C	1	1	0.5	0.5	1	4
Vogt et al. [180]	2007	C	1	0.5	1	1	1	4.5
Jim et al. [172]	2007	C	0.5	1	0.5	1	1	4
Martin and Lam [85]	2008	C	0.5	1	0.5	1	1	4
McAllister et al. [90]	2008	C	1	0.5	0.5	1	1	4
Balzarotti et al. [128]	2008	C	0.5	1	0.5	1	1	4
Wassermann and Su [82]	2008	C	1	1	0.5	1	1	4.5
Johns et al. [145]	2008	C	1	1	0.5	1	1	4.5
Bisht and Venkatakrishnan [193]	2008	C	1	1	0.5	0.5	1	4
Iha and Doi [130]	2009	C	1	1	0.5	1	1	4.5
Kiezun et al. [122]	2009	C	0.5	1	0.5	1	1	4
Louw and Venkatakrishnan [129]	2009	C	1	1	1	1	1	5
Kirda et al. [171]	2009	J	1	1	0.5	1	1	4.5
Sun et al. [146]	2009	C	1	1	0.5	1	1	4.5
Nadji et al. [185]	2009	C	1	0.5	1	1	1	4.5
Barth et al. [182]	2009	C	1	1	0.5	1	1	4.5
Wurzinger et al. [162]	2009	C	1	1	0.5	1	1	4.5
Gebre et al. [167]	2010	C	1	1	0.5	0.5	1	4
Bates et al. [194]	2010	C	1	1	0.5	1	1	4.5
Stamm et al. [174]	2010	C	0.5	1	0.5	1	1	4
Athanasopoulos et al. [186]	2010	C	1	1	1	0	1	4
Weinberger et al. [40]	2011	C	1	1	1	1	1	5
Samuel et al. [68]	2011	C	1	1	0.5	1	1	4.5
Hooimeijer et al. [71]	2011	C	1	1	0.5	1	1	4.5
Mui and Frankl [179]	2011	C	0.5	1	0.5	1	1	4
Wang et al. [80]	2011	C	1	1	0.5	0.5	1	4
Shariar and Zulkernine [187]	2011	C	1	1	0.5	0.5	1	4
Avancini and Ceccato [111]	2011	C	1	1	0.5	0.5	1	4
Barua et al. [183]	2011	C	1	1	1	1	1	5
Scholte et al. [50]	2012	C	0.5	1	1	1	1	4.5
Shar and Tan [72]	2012	C	1	1	0.5	0.5	1	4
Shar and Tan [133]	2012	J	1	1	1	1	1	5
Nunan et al. [149]	2012	C	1	1	1	1	1	5
Van Acker et al. [127]	2012	C	1	1	1	1	1	5
Van-Gundy and Chen [190]	2012	J	1	1	1	1	1	5
Cao et al. [192]	2012	C	1	1	0.5	1	1	4.5
Scholte et al. [132]	2012	C	0.5	1	0.5	1	1	4
Pelizzi and Sekar [195]	2012	C	1	1	1	1	1	5
Sundareswaran and Squicciarini [147]	2012	C	1	1	0.5	1	1	4.5
Lekies et al. [119]	2013	C	1	1	1	1	1	5
Faghani and Nguyen [41]	2013	J	1	1	1	1	1	5
Avancini and Ceccato [98]	2013	C	1	1	0.5	1	1	4.5
Avancini and Ceccato [51]	2013	J	1	1	1	1	1	5
Doupé et al. [175]	2013	C	1	1	1	1	1	5
Das et al. [148]	2013	J	1	0.5	0.5	1	1	4
Tripp et al. [97]	2013	C	1	1	0.5	1	1	4.5
Duchène et al. [91]	2013	C	1	1	1	1	1	5
Heiderich et al. [35]	2013	C	1	1	0.5	1	1	4.5
Shar and Tan [74]	2013	J	0.5	1	1	1	1	4.5
Vernotte et al. [73]	2014	C	1	0.5	0.5	1	1	4
Rocha and Souto [96]	2014	C	1	0.5	0.5	1	1	4
Duchène et al. [92]	2014	C	1	1	1	0	1	4
Wang et al. [134]	2014	C	1	0.5	0.5	1	1	4
Stock et al. [181]	2014	C	1	1	1	1	1	5
Johns [173]	2014	J	1	1	0.5	0.5	1	4
Javed and Schwenk [168]	2014	C	1	1	0.5	0.5	1	4
Bozic et al. [105]	2015	C	1	1	1	1	1	5
Fazzini et al. [176]	2015	C	1	1	0.5	1	1	4.5
Parameshwaran et al. [131]	2015	C	1	1	0.5	1	1	4.5
Bozic et al. [52]	2015	C	1	1	1	1	1	5
Stock et al. [53]	2015	C	1	1	1	1	1	5
Zhang et al. [39]	2015	J	1	1	0.5	1	1	4.5
Simos and Kleine [106]	2016	C	1	1	0.5	0.5	1	4
Gupta and Gupta [9]	2016	C	1	0.5	0.5	1	1	4
Weichselbaum et al. [54]	2016	C	1	1	1	1	1	5
Pan et al. [177]	2016	C	1	1	1	1	1	5
Pan and Mao [124]	2016	C	1	1	0.5	1	1	4.5
Mitropoulos et al. [163]	2016	J	1	1	0.5	1	1	4.5
Steinhauser and Gauthier [81]	2016	C	1	1	1	0	1	4
Ahmed and Ali [109]	2016	J	1	1	0.5	1	1	4.5
Lin and Barcelo [55]	2016	C	1	1	1	1	1	5
Yan and Qiao [83]	2016	C	1	1	0.5	0.5	1	4

(continued on next page)

Table 13 (*continued*).

Ref.	Year	Type	QA1	QA2	QA3	QA4	QA5	Total QA Score
Pan et al. [88]	2016	J	1	1	1	1	0	4
Gupta et al [75]	2016	C	1	1	1	0	1	4
Bazzoli et al. [56]	2016	C	1	1	1	0	1	4
Gupta and Gupta [196]	2016	J	1	1	0.5	1	1	4.5
Gupta and Gupta [78]	2016	J	1	1	0.5	1	1	4.5
Goswami et al. [150]	2017	J	1	0.5	0.5	1	1	4
Lekies et al. [42]	2017	C	1	1	1	1	1	5
Mohammadi et al. [99]	2017	C	1	1	0.5	1	1	4.5
Pan and Mao [125]	2017	C	1	1	0.5	1	1	4.5
Marashdih and Zaaba [46]	2017	C	1	1	0.5	0.5	1	4
Heiderich et al. [70]	2017	C	1	1	0.5	1	1	4.5
Gupta and Gupta [188]	2017	J	1	1	0.5	0.5	1	4
Marashdih et al. [110]	2017	J	1	1	0.5	0.5	1	4
Rathore et al. [135]	2017	J	1	1	0.5	0.5	1	4
Ayeni et al. [102]	2018	J	1	1	0.5	1	1	4.5
Gupta et al. [10]	2018	J	1	1	1	1	1	5
Mereani and Howe [136]	2018	C	1	1	1	0.5	1	4.5
Melicher et al. [36]	2018	C	1	1	1	1	1	5
Gupta and Gupta [164]	2018	J	1	1	1	0.5	1	4.5
Wang et al. [121]	2018	J	1	0.5	1	1	1	4.5
Yamazaki et al. [64]	2018	C	1	1	1	1	0	4
Lalia and Sarah [197]	2018	C	1	1	0.5	0.5	1	4
Gupta and Gupta [11]	2018	J	1	1	1	1	1	5
Gupta et al. [45]	2019	J	1	1	0.5	0.5	1	4
Garn et al. [107]	2019	C	1	0.5	0.5	1	1	4
Chaudhary et al. [49]	2019	J	1	1	0.5	0.5	1	4
Lv et al. [95]	2019	C	1	1	0.5	1	1	4.5
Zhou and Wang [139]	2019	J	1	1	1	1	1	5
Zhang et al. [137]	2019	C	1	1	0.5	1	1	4.5
Steinhauser and Tuma [87]	2019	J	1	1	1	1	1	5
Steffens et al. [37]	2019	C	1	1	1	1	1	5
Wijayarathna and Arachchilage [57]	2019	C	1	1	0.5	1	1	4.5
Chaliasos et al. [38]	2019	C	1	1	0.5	1	1	4.5
Mokbal et al. [151]	2019	J	1	1	1	1	1	5
Iqbal et al. [178]	2019	C	1	1	1	1	0	4
Simos et al. [104]	2019	C	1	1	0.5	0.5	1	4
Fang et al. [140]	2019	J	1	1	0.5	1	1	4.5
Musch et al. [65]	2019	C	1	1	0.5	1	1	4.5
Niakanlahiji and Jafarian [191]	2019	J	1	1	0.5	1	1	4.5
Kadhim and Gaata [157]	2020	J	1	1	0.5	0.5	1	4
Zhang et al. [141]	2020	J	1	1	1	1	1	5
Li et al. [76]	2020	J	1	1	0.5	1	1	4.5
Mokbal et al. [144]	2020	J	1	1	1	1	1	5
Steinhauser and Tuma [100]	2020	J	1	1	1	0	1	4
Gupta et al. [189]	2020	J	1	1	0.5	1	1	4.5
Schuckert et al. [58]	2020	C	1	1	0.5	1	1	4.5
Nagarjun and Ahamad [154]	2020	J	1	1	1	0.5	1	4.5
Xu et al. [48]	2020	J	1	1	1	1	1	5
Fang et al. [158]	2020	J	1	1	0.5	1	1	4.5
Buyukkayhan et al. [59]	2020	C	1	1	0.5	1	1	4.5
Li et al. [138]	2020	C	1	1	0.5	0.5	1	4
Bui et al. [8]	2020	C	1	1	0.5	1	1	4.5
Chaudhary et al. [165]	2020	C	1	1	1	0	1	4
Talib and Doh [60]	2021	J	1	1	1	0.5	1	4.5
Eriksson et al. [93]	2021	C	1	0.5	1	1	1	4.5
Garn et al. [108]	2021	C	1	1	1	1	1	5
Malviya et al. [155]	2021	J	1	1	1	1	1	5
Caturano et al. [114]	2021	J	1	1	0.5	1	1	4.5
Chaudhary et al. [159]	2021	J	1	1	1	1	1	5
Frempong et al. [116]	2021	C	1	0.5	0.5	1	1	4
Leithner et al. [44]	2021	J	1	1	0.5	1	1	4.5
Wang et al. [43]	2021	C	1	1	0.5	1	1	4.5
Tariq et al. [47]	2021	J	1	1	1	1	1	5
Mereani and Howe [152]	2021	C	1	1	0.5	0.5	1	4
Bensalim et al. [120]	2021	C	1	1	1	0	1	4
Melicher et al. [101]	2021	C	1	1	1	0	1	4
Mokbal et al. [156]	2021	J	1	1	1	1	1	5
Pazos et al. [166]	2021	C	1	1	0.5	1	1	4.5
Wang et al. [142]	2022	J	1	0.5	1	1	1	4.5
Mokbal et al. [153]	2022	J	1	1	0.5	1	1	4.5
Liu et al. [160]	2022	J	1	1	0.5	1	1	4.5
Maurel et al. [77]	2022	J	1	1	0.5	1	1	4.5
Korac et al. [61]	2022	J	1	0.5	0.5	1	1	4
Su et al. [84]	2022	C	1	1	1	1	1	5
Krishnan et al. [198]	2022	J	1	0.5	0.5	1	1	4
Liu et al. [113]	2022	J	1	0.5	1	1	1	4.5

(*continued on next page*)

Table 13 (*continued*).

Ref.	Year	Type	QA1	QA2	QA3	QA4	QA5	Total QA Score
Pan et al. [161]	2022	J	1	0.5	1	0.5	1	4
Shar et al. [62]	2022	C	1	1	0.5	1	1	4.5
Chen et al. [143]	2022	J	1	1	1	1	1	5
Foley and Maffeis. [117]	2022	C	1	1	0.5	1	1	4.5
Lee et al. [118]	2022	C	1	1	1	1	1	5
Chaudhary et al. [6]	2022	J	1	0.5	0.5	1	1	4

References

[1] G. Rossi, O. Pastor, D. Schwabe, L. Olsina, Web Engineering: Modelling and Implementing Web Applications, in: Human-Computer Interaction Series, Springer, 2008, http://dx.doi.org/10.1007/978-1-84628-923-1.

[2] Y. Sadqi, Y. Maleh, A systematic review and taxonomy of web applications threats, Inform. Secur. J.: A Glob. Perspect. 31 (1) (2022) 1–27, http://dx.doi.org/10.1080/19393555.2020.1853855.

[3] J. Grossman, R. Hansen, P.D. Petkov, A. Rager, XSS attacks: Cross-site scripting exploits and defense, syngress, 2007.

[4] OWASP, The open web application security project, 2021, https://owasp.org/www-project-top-ten/.

[5] Accountix, Acunetix web application vulnerability-report, 2021, https://www.acunetix.com/white-papers/acunetix-web-application-vulnerability-report-2021/.

[6] P. Chaudhary, B.B. Gupta, A.K. Singh, Securing heterogeneous embedded devices against XSS attack in intelligent IoT system, Comput. Secur. 118 (2022) 102710, http://dx.doi.org/10.1016/j.cose.2022.102710.

[7] G. Shivi, B. Niyati, Comparative analysis of android and iOS from security viewpoint, Comp. Sci. Rev. 40 (2021) 100372, http://dx.doi.org/10.1016/j.cosrev.2021.100372.

[8] T. Bui, S. Rao, M. Antikainen, T. Aura, Xss vulnerabilities in cloud-application add-ons, in: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, ASIACCS, ACM, Taipei, Taiwan, 2020, pp. 610–621, http://dx.doi.org/10.1145/3320269.3384744.

[9] S. Gupta, B.B. Gupta, An infrastructure-based framework for the alleviation of Javascript worms from osn in mobile cloud platforms, in: International Conference on Network and System Security, NSS, Springer, Taipei, Taiwan, 2016, pp. 98–109, http://dx.doi.org/10.1007/978-3-319-46298-1_7.

[10] S. Gupta, B. Gupta, P. Chaudhary, Hunting for dom-based xss vulnerabilities in mobile cloud-based online social network, Future Gener. Comput. Syst. 79 (2018) 319–336, http://dx.doi.org/10.1016/j.future.2017.05.038.

[11] S. Gupta, B. Gupta, Xss-secure as a service for the platforms of online social network-based multimedia web applications in cloud, Multimedia Tools Appl. 77 (4) (2018) 4829–4861, http://dx.doi.org/10.1007/s11042-016-3735-1.

[12] R. Kumar, R. Goyal, On cloud security requirements, threats, vulnerabilities and countermeasures: A survey, Comp. Sci. Rev. 33 (2019) 1–48, http://dx.doi.org/10.1016/j.cosrev.2019.05.002.

[13] S. Gupta, B.B. Gupta, Evaluation and monitoring of xss defensive solutions: A survey, open research issues and future directions, J. Ambient Intell. Humaniz. Comput. 10 (11) (2019) 4377–4405, http://dx.doi.org/10.1007/s12652-018-1118-3.

[14] M. Liu, B. Zhang, W. Chen, X. Zhang, A survey of exploitation and detection methods of xss vulnerabilities, IEEE Access 7 (2019) 182004–182016, http://dx.doi.org/10.1109/ACCESS.2019.2960449.

[15] G.E. Rodríguez, J.G. Torres, P. Flores, D.E. Benavides, Cross-site scripting (xss) attacks and mitigation: A survey, Comput. Netw. 166 (2020) 106960, http://dx.doi.org/10.1016/j.comnet.2019.106960.

[16] B.A. Kitchenham, D. Budgen, P. Brereton, Evidence-Based Software Engineering and Systematic Reviews, Chapman & Hall/CRC, 2015.

[17] V.K. Malviya, S. Saurav, A. Gupta, On security issues in web applications through cross site scripting (xss), in: Proceedings of the 20th Asia-Pacific Software Engineering Conference, APSEC, IEEE, Bangkok, Thailand, 2013, pp. 583–588, http://dx.doi.org/10.1109/APSEC.2013.85.

[18] I. Hydara, A.B.M. Sultan, H. Zulzalil, N. Admodisastro, Current state of research on cross-site scripting (xss) – a systematic literature review, Inf. Softw. Technol. 58 (2015) 170–186, http://dx.doi.org/10.1016/j.infsof.2014.07.010.

[19] V. Nithya, S.L. Pandian, C. Malarvizhi, A survey on detection and prevention of cross-site scripting attack, Int. J. Secur. Appl. 9 (3) (2015) 139–152, http://dx.doi.org/10.14257/ijsia.2015.9.3.14.

[20] G. Deepa, P.S. Thilagam, Securing web applications from injection and logic vulnerabilities: Approaches and challenges, Inf. Softw. Technol. 74 (2016) 160–180, http://dx.doi.org/10.1016/j.infsof.2016.02.005.

[21] S. Gupta, B.B. Gupta, Cross-site scripting (xss) attacks and defense mechanisms: Classification and state-of-the-art, Int. J. Syst. Assur. Eng. Manag. 8 (1) (2017) 512–530, http://dx.doi.org/10.1007/s13198-015-0376-0.

[22] P. Chaudhary, B. Gupta, Plague of cross-site scripting on web applications: A review, taxonomy and challenges, Int. J. Web Based Commun. 14 (1) (2018) 64–93, http://dx.doi.org/10.1504/IJWBC.2018.090916.

[23] U. Sarmah, D. Bhattacharyya, J. Kalita, A survey of detection methods for xss attacks, J. Netw. Comput. Appl. 118 (2018) 113–143, http://dx.doi.org/10.1016/j.jnca.2018.06.004.

[24] M. Kuhrmann, D.M. Fernández, M. Daneva, On the pragmatic design of literature studies in software engineering: An experience-based guideline, Empir. Softw. Eng. 22 (6) (2017) 2852–2891, http://dx.doi.org/10.1007/s10664-016-9492-y.

[25] M. Petticrew, H. Roberts, Systematic Reviews in the Social Sciences: A Practical Guide, John Wiley & Sons, Ltd, 2006.

[26] A. Hannousse, Searching relevant papers for software engineering secondary studies: Semantic scholar coverage and identification role, IET Softw. 15 (2019) 126–146, http://dx.doi.org/10.1049/sfw2.12011.

[27] B. Cartaxo, G. Pinto, S. Gustavo, S. Soares, Rapid reviews in software engineering, Contemp. Emp. Methods Softw. Eng. (2020) 357–384, http://dx.doi.org/10.1007/978-3-030-32489-6_1.

[28] R. Pranckute, Web of science (WoS) and scopus: The titans of bibliographic information in today's academic world, Publications 9 (2021) 12, http://dx.doi.org/10.3390/publications9010012.

[29] C. Wohlin, Second-generation systematic literature studies using snowballing, in: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, EASE, ACM, Limerick, Ireland, 2016, pp. 1–6, http://dx.doi.org/10.1145/2915970.2916006.

[30] Y. Zhou, He Zhang, X. Huang, S. Yang, B. Song, A. Muhammad, H. Tang, Quality assessment of systematic reviews in software engineering: A tertiary study, in: Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, EASE, ACM, Nanjing, China, 2015, pp. 1–14, http://dx.doi.org/10.1145/2745802.2745815.

[31] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE, ACM, Bari, Italy, 2008, pp. 68–77.

[32] D.S. Cruzes, T. Dybå, Research synthesis in software engineering: A tertiary study, Inf. Softw. Technol. 53 (5) (2011) 440–455, http://dx.doi.org/10.1016/j.infsof.2011.01.004.

[33] M. Aria, C. Cuccurullo, Bibliometrix: An R-tool for comprehensive science mapping analysis, J. Informetr. 11 (4) (2017) 959–975, http://dx.doi.org/10.1016/j.joi.2017.08.007.

[34] J.Y. Halper, D.C. Parkes, Journals for certification, conferences for rapid dissemination, Commun. ACM 54 (8) (2011) 36–38, http://dx.doi.org/10.1145/1978542.1978555.

[35] M. Heiderich, J. Schwenk, T. Frosch, J. Magazinius, E.Z. Yang, Mxss attacks: Attacking well-secured web-applications by using innerhtml mutations, in: Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS, ACM, Berlin, Germany, 2013, pp. 777–788, http://dx.doi.org/10.1145/2508859.2516723.

[36] W. Melicher, A. Das, M. Sharif, L. Bauer, L. Jia, Riding out domsday: Towards detecting and preventing dom cross-site scripting, in: Proceedings of the Network and Distributed System Security Symposium, NDSS, The Internet Society, San Diego, California, USA, 2018, pp. 1–15, http://dx.doi.org/10.14722/ndss.2018.23309.

[37] M. Steffens, C. Rossow, M. Johns, B. Stock, Don't trust the locals: Investigating the prevalence of persistent client-side cross-site scripting in the wild, in: Proceedings of the Network and Distributed System Security Symposium, NDSS, The Internet Society, San Diego, California, USA, 2019, pp. 1–15, http://dx.doi.org/10.14722/ndss.2019.23009.

[38] S. Chaliasos, G. Metaxopoulos, G. Argyros, D. Mitropoulos, Mime artist: Bypassing whitelisting for the web with Javascript mimicry attacks, in: Proceedings of the European Symposium on Research in Computer Security, ESORICS, Springer, Luxembourg City, Luxembourg, 2019, pp. 565–585, http://dx.doi.org/10.1007/978-3-030-29962-0_27.

[39] Y. Zhang, Q. Liu, Q. Luo, X. Wang, Xas: Cross-api scripting attacks in social ecosystems, Sci. China Inf. Sci. 58 (1) (2015) 1–14, http://dx.doi.org/10.1007/s11432-014-5145-1.

[40] J. Weinberger, P. Saxena, D. Akhawe, M. Finifter, R. Shin, D. Song, A systematic analysis of xss sanitization in web application frameworks, in: Proceedings of the European Symposium on Research in Computer Security, ESORICS, Springer, Leuven, Belgium, 2011, pp. 150–171, http://dx.doi.org/10.1007/978-3-642-23822-2_9.

[41] M. Faghani, U. Nguyen, A study of xss worm propagation and detection mechanisms in online social networks, IEEE Trans. Inf. Forensics Secur. 8 (11) (2013) 1815–1826, http://dx.doi.org/10.1109/TIFS.2013.2280884.

[42] S. Lekies, K. Kotowicz, S. Grob, E. Nava, M. Johns, Code-reuse attacks for theweb: Breaking cross-site scripting mitigations via script gadgets, in: Proceedings of the ACM Conference on Computer and Communications Security, CCS, ACM, Dallas, Texas, USA, 2017, pp. 1709–1723, http://dx.doi.org/10.1145/3133956.3134091.

[43] P. Wang, J. Bangert, C. Kern, If it's not secure, it should not compile: Preventing dom-based xss in large-scale web development with api hardening, in: Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering, ICSE, ICSE, IEEE, Madrid, Spain, 2021, pp. 1360–1372, http://dx.doi.org/10.1109/ICSE43902.2021.00123.

[44] M. Leithner, B. Garn, D.E. Simos, Hydra: Feedback-driven black-box exploitation of injection vulnerabilities, Inf. Softw. Technol. 140 (2021) 106703, http://dx.doi.org/10.1016/j.infsof.2021.106703.

[45] S. Gupta, B. Gupta, P. Chaudhary, A client–server Javascript code rewriting-based framework to detect the xss worms from online social network, Concurr. Comput.: Pract. Exper. 31 (21) (2019) 1–20, http://dx.doi.org/10.1002/cpe.4646.

[46] A. Marashdih, Z. Zaaba, Detection and removing cross site scripting vulnerability in php web application, in: Proceedings of the 2017 International Conference on Promising Electronic Technologies, ICPET, IEEE, Deir El-Balah, Palestine, 2017, pp. 26–31, http://dx.doi.org/10.1109/ICPET.2017.11.

[47] I. Tariq, M. Sindhu, R. Abbasi, A. Khattak, O. Maqbool, G. Siddiqui, Resolving cross-site scripting attacks through genetic algorithm and reinforcement learning, Expert Syst. Appl. 168 (2021) 114386, http://dx.doi.org/10.1016/j.eswa.2020.114386.

[48] G. Xu, X. Xie, S. Huang, J. Zhang, L. Pan, W. Lou, K. Liang, Jscsp: A novel policy-based xss defense mechanism for browsers, IEEE Trans. Dependable Secure Comput. (2020) 1–17, http://dx.doi.org/10.1109/TDSC.2020.3009472.

[49] P. Chaudhary, B. Gupta, S. Gupta, A framework for preserving the privacy of online users against xss worms on online social network, Int. J. Inform. Technol. Web Eng. 14 (1) (2019) 85–111, http://dx.doi.org/10.4018/IJITWE.2019010105.

[50] T. Scholte, W. Robertson, D. Balzarotti, E. Kirda, An empirical analysis of input validation mechanisms in web applications and languages, in: Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC, ACM, Trento, Italy, 2012, pp. 1419–1426, http://dx.doi.org/10.1145/2245276.2232004.

[51] A. Avancini, M. Ceccato, Comparison and integration of genetic algorithms and dynamic symbolic execution for security testing of cross-site scripting vulnerabilities, Inf. Softw. Technol. 55 (12) (2013) 2209–2222, http://dx.doi.org/10.1016/j.infsof.2013.08.001.

[52] J. Bozic, B. Garn, D.E. Simos, F. Wotawa, Evaluation of the ipo-family algorithms for test case generation in web security testing, in: Proceedings of the 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops, ICSTW, IEEE, Graz, Austria, 2015, pp. 1–10, http://dx.doi.org/10.1109/ICSTW.2015.7107436.

[53] B. Stock, S. Pfistner, B. Kaiser, S. Lekies, M. Johns, From facepalm to brain bender: Exploring client-side cross-site scripting, in: Proceedings of the ACM Conference on Computer and Communications Security, CCS, ACM, Denver, Colorado, USA, 2015, pp. 1419–1430, http://dx.doi.org/10.1145/2810103.2813625.

[54] L. Weichselbaum, M. Spagnuolo, S. Lekies, A. Janc, Csp is dead, long live csp! on the insecurity of whitelists and the future of content security policy, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS, ACM, Vienna, Austria, 2016, pp. 1376–1387, http://dx.doi.org/10.1145/2976749.2978363.

[55] A. Lin, P. Barceló, String solving with word equations and transducers: Towards a logic for analysing mutation xss, in: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL, ACM, St. Petersburg, FL, USA, 2016, pp. 123–136, http://dx.doi.org/10.1145/2837614.2837641.

[56] E. Bazzoli, C. Criscione, F. Maggi, S. Zanero, Xss peeker: Dissecting the xss exploitation techniques and fuzzing mechanisms of blackbox web application scanners, in: Proceedings of the 31st IFIP International Conference on ICT Systems Security and Privacy Protection, SEC, Springer, Ghent, Belgium, 2016, pp. 243–258, http://dx.doi.org/10.1007/978-3-319-33630-5_17.

[57] C. Wijayarathna, N. Arachchilage, Fighting against xss attacks: A usability evaluation of owasp esapi output encoding, in: Proceedings of the 52th Annual Hawaii International Conference on System Sciences, HICSS, AIS Electronic Library, Grand Wailea, Maui, Hawaii, USA, 2019, pp. 7302–7311, http://dx.doi.org/10.24251/HICSS.2019.877.

[58] F. Schuckert, B. Katt, H. Langweg, Difficult xss code patterns for static code analysis tools, in: Proceedings of the ESORICS 2019 International Workshops, IOSec, MSTEC, and FINSEC, ESORICS, Springer, Luxembourg City, Luxembourg, 2020, pp. 123–139, http://dx.doi.org/10.1007/978-3-030-42051-2_9.

[59] A. Buyukkayhan, C. Gemicioglu, T. Lauinger, A. Oprea, W. Robertson, E. Kirda, What's in an exploit? an empirical analysis of reflected server xss exploitation techniques, in: Proceedings of the 23rd International Symposium on Research in Attacks, Intrusions and Defenses, RAID, USENIX, Donostia/San Sebastian, Spain, 2020, pp. 107–120.

[60] N.A.A. Talib, K.-G. Doh, Assessment of dynamic open-source cross-site scripting filters for web application, KSII Trans. Internet Inform. Syst. (TIIS) 15 (10) (2021) 3750–3770, http://dx.doi.org/10.3837/tiis.2021.10.015.

[61] D. Korac, B. Damjanovic, D. Simic, K.K.R. Choo, A hybrid XSS attack (HYXSSA) based on fusion approach: Challenges, threats and implications in cybersecurity, J. King Saud Univ. – Comput. Inform. Sci. 34 (2022) 9284–9300, http://dx.doi.org/10.1016/j.jksuci.2022.09.008.

[62] L.K. Shar, C.M. Poskitt, K.J. Shim, L.Y.L. Wong, H. Pan, Y. Fang, C. Huang, W. Guo, X. Wan, XSS for the masses: Integrating security in a web programming course using a security scanner, in: In Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, ACM, Dublin, Ireland, 2022, pp. 463–469, http://dx.doi.org/10.1145/3502718.3524795.

[63] OWASP, Types of cross-site scripting. https://owasp.org/www-community/Types_of_Cross-Site_Scripting#Types_of_Cross-Site_Scripting.

[64] K. Yamazaki, D. Kotani, Y. Okabe, Xilara: An xss filter based on html template restoration, in: Proceedings of the 14th International Conference on Security and Privacy in Communication Systems, SecureComm, Springer, Singapore, Singapore, 2018, pp. 332–351, http://dx.doi.org/10.1007/978-3-030-01704-0_18.

[65] M. Musch, M. Steffens, S. Roth, B. Stock, M. Johns, Scriptprotect: Mitigating unsafe third-party Javascript practices, in: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, ASIACCS, ACM, Auckland, New Zealand, 2019, pp. 391–402, http://dx.doi.org/10.1145/3321705.3329841.

[66] A. Klein, Dom based cross site scripting or xss of the third kind: A look at an overlooked flavor of xss. http://www.webappsec.org/projects/articles/071105.html.

[67] D. Wagner, P. Soto, Mimicry attacks on host-based intrusion detection systems, in: Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS, ACM, Washington, DC, USA, 2002, pp. 255–264, http://dx.doi.org/10.1145/586110.586145.

[68] M. Samuel, P. Saxena, D. Song, Context-sensitive auto-sanitization in web templating languages using type qualifiers, in: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS, ACM, Chicago, Illinois, USA, 2011, pp. 587–600, http://dx.doi.org/10.1145/2046707.2046775.

[69] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, S.-Y. Kuo, Securing web application code by static analysis and runtime protection, in: Proceedings of the 13th International Conference on World Wide Web, WWW, ACM, New York, NY, USA, 2004, pp. 40–52, http://dx.doi.org/10.1145/988672.988679.

[70] M. Heiderich, C. Späth, J. Schwenk, Dompurify: Client-side protection against xss and markup injection, in: Proceedings of the 2017 European Symposium on Research in Computer Security, ESORICS, Springer, Oslo, Norway, 2017, pp. 116–134, http://dx.doi.org/10.1007/978-3-319-66399-9_7.

[71] P. Hooimeijer, B. Livshits, D. Molnar, P. Saxena, M. Veanes, Fast and precise sanitizer analysis with $\{BEK\}$, in: Proceedings of the 20th USENIX Security Symposium, USENIX, USENIX, San Francisco, CA, USA, 2011, pp. 1–16.

[72] L. Shar, H. Tan, Auditing the xss defence features implemented in web application programs, IET Softw. 6 (4) (2012) 377–390, http://dx.doi.org/10.1049/iet-sen.2011.0084.

[73] A. Vernotte, F. Dadeau, F. Lebeau, B. Legeard, F. Peureux, F. Piat, Efficient detection of multi-step cross-site scripting vulnerabilities, in: Proceedings of the International Conference on Information Systems Security, ICISS, Springer, Hyderabad, India, 2014, pp. 358–377, http://dx.doi.org/10.1007/978-3-319-13841-1_20.

[74] L. Shar, H. Tan, Predicting sql injection and cross site scripting vulnerabilities through mining input sanitization patterns, Inf. Softw. Technol. 55 (10) (2013) 1767–1780, http://dx.doi.org/10.1016/j.infsof.2013.04.002.

[75] M. Gupta, M. Govil, G. Singh, Text-mining based predictive model to detect xss vulnerable files in web applications, in: Poroceedings of the 12th IEEE International Conference Electronics, Energy, Environment, Communication, Computer, Control, INDICON, IEEE, New Delhi, India, 2016, pp. 1–6, http://dx.doi.org/10.1109/INDICON.2015.7443332.

[76] C. Li, Y. Wang, C. Miao, C. Huang, Cross-site scripting guardian: A static xss detector based on data stream input–output association mining, Appl. Sci. (Switzerland) 10 (14) (2020) 1–20, http://dx.doi.org/10.3390/app10144740.

[77] H. Maurel, S. Vidal, T. Rezk, Statically identifying xss using deep learning, Sci. Comput. Programm. 219 (2022) 102810, http://dx.doi.org/10.1016/j.scico.2022.102810.

[78] S. Gupta, B. Gupta, Xss-safe: A server-side approach to detect and mitigate cross-site scripting (xss) attacks in Javascript code, Arab. J. Sci. Eng. 41 (3) (2016) 897–920, http://dx.doi.org/10.1007/s13369-015-1891-7.

[79] N. Jovanovic, C. Kruegel, E. Kirda, Pixy: A static analysis tool for detecting web application vulnerabilities, in: Proceedings of the 2006 IEEE Symposium on Security and Privacy, SP, IEEE, Berkeley/Oakland, CA, USA, 2006, pp. 1–6, http://dx.doi.org/10.1109/SP.2006.29.

[80] Y. Wang, Z. Li, T. Guo, Program slicing stored xss bugs in web application, in: Proceedings of the 5th International Conference on Theoretical Aspects of Software Engineering, TASE, IEEE, Xi'an, China, 2011, pp. 191–194, http://dx.doi.org/10.1109/TASE.2011.43.

[81] A. Steinhauser, F. Gauthier, Jspchecker: Static detection of context-sensitive cross-site scripting flaws in legacy web applications, in: Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, PLAS, ACM, Vienna, Austria, 2016, pp. 57–68, http://dx.doi.org/10.1145/2993600.2993606.

[82] G. Wassermann, Z. Su, Static detection of cross-site scripting vulnerabilities, in: Proceedings of the ACM/IEEE 30th International Conference on Software Engineering, ICSE, ACM, Leipzig, Germany, 2008, pp. 171–180, http://dx.doi.org/10.1145/1368088.1368112.

[83] F. Yan, T. Qiao, Study on the detection of cross-site scripting vulnerabilities based on reverse code audit, in: Proceedings of the 17th International Conference on Intelligent Data Engineering and Automated Learning, IDEAL, Springer, Yangzhou, China, 2016, pp. 154–163, http://dx.doi.org/10.1007/978-3-319-46257-8_17.

[84] H. Su, L. Xu, H. Chao, F. Li, Z. Yuan, J. Zhou, W. Huo, A sanitizer-centric analysis to detect cross-site scripting in PHP programs, in: In Proceedings of the International Symposium on Software Reliability Engineering, ISSRE, IEEE, Charlotte, North Carolina, USA, 2022, pp. 355–365, http://dx.doi.org/10.1109/ISSRE55969.2022.00042.

[85] M. Martin, M. Lam, Automatic generation of xss and sql injection attacks with goal-directed model checking, in: Proceedings of the 17th USENIX Security Symposium, USENIX, USENIX, San Jose, CA, USA, 2008, pp. 31–43.

[86] M. Martin, B. Livshits, M.S. Lam, Finding application errors and security flaws using pql: A program query language, in: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA, ACM, San Diego, CA, USA, 2005, pp. 365–383, http://dx.doi.org/10.1145/1094811.1094840.

[87] A. Steinhauser, P. Tůma, Djangochecker: Applying extended taint tracking and server side parsing for detection of context-sensitive xss flaws, Softw. - Pract. Exper. 49 (1) (2019) 130–148, http://dx.doi.org/10.1002/spe.2649.

[88] J. Pan, X. Mao, W. Li, Taint inference for cross-site scripting in context of url rewriting and html sanitization, ETRI J. 38 (2) (2016) 376–386, http://dx.doi.org/10.4218/etrij.16.0115.0570.

[89] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, USA, 1997.

[90] S. McAllister, E. Kirda, C. Kruegel, Leveraging user interactions for in-depth testing of web applications, in: International Workshop on Recent Advances in Intrusion Detection, RAID, Springer, Cambridge, MA, USA, 2008, pp. 191–210, http://dx.doi.org/10.1007/978-3-540-87403-4_11.

[91] F. Duchene, S. Rawat, J.-L. Richier, R. Groz, Ligre: Reverse-engineering of control and data flow models for black-box xss detection, in: Proceedings of the Working Conference on Reverse Engineering, WCRE, WCRE, IEEE, Koblenz, Germany, 2013, pp. 252–261, http://dx.doi.org/10.1109/WCRE.2013.6671300.

[92] F. Duchene, S. Rawat, J.-L. Richier, R. Groz, Kameleonfuzz: Evolutionary fuzzing for black-box xss detection, in: Proceedings of the 4th ACM Conference on Data and Application Security and Privacy, CODASPY, ACM, San Antonio, Texas, USA, 2014, pp. 37–48, http://dx.doi.org/10.1145/2557547.2557550.

[93] B. Eriksson, G. Pellegrino, A. Sabelfeld, Black widow: Blackbox data-driven web scanning, in: Proceedings of the 2021 IEEE Symposium on Security and Privacy, SP, IEEE, San Francisco, CA, USA, 2021, pp. 1125–1142, http://dx.doi.org/10.1109/SP40001.2021.00022.

[94] R. Huang, W. Sun, Y. Xu, H. Chen, D. Towey, X. Xia, A survey on adaptive random testing, IEEE Trans. Softw. Eng. 47 (10) (2021) 2052–2083, http://dx.doi.org/10.1109/TSE.2019.2942921.

[95] C. Lv, L. Zhang, F. Zeng, J. Zhang, Adaptive random testing for xss vulnerability, in: Proceedings of the 26th Asia-Pacific Software Engineering Conference, APSEC, IEEE, Putrajaya, Malaysia, 2019, pp. 63–69, http://dx.doi.org/10.1109/APSEC48747.2019.00018.

[96] T. Rocha, E. Souto, Etssdetector: A tool to automatically detect cross-site scripting vulnerabilities, in: Proceedings of the 2014 IEEE 13th International Symposium on Network Computing and Applications, NCA, IEEE, Cambridge, MA, USA, 2014, pp. 306–309, http://dx.doi.org/10.1109/NCA.2014.53.

[97] O. Tripp, O. Weisman, L. Guy, Finding your way in the testing jungle: A learning approach to web security testing, in: Proceedings of the 2013 International Symposium on Software Testing and Analysis, ISSTA, ACM, Lugano, Switzerland, 2013, pp. 347–357, http://dx.doi.org/10.1145/2483760.2483776.

[98] A. Avancini, M. Ceccato, Circe: A grammar-based oracle for testing cross-site scripting in web applications, in: Proceedings of the 2013 20th Working Conference on Reverse Engineering, WCRE, IEEE, Koblenz, Germany, 2013, pp. 262–271, http://dx.doi.org/10.1109/WCRE.2013.6671301.

[99] M. Mohammadi, B. Chu, H. Lipford, Detecting cross-site scripting vulnerabilities through automated unit testing, in: Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security, QRS, IEEE, Prague, Czech Republic, 2017, pp. 364–373, http://dx.doi.org/10.1109/QRS.2017.46.

[100] A. Steinhauser, P. Tůma, Database traffic interception for graybox detection of stored and context-sensitive xss, Digit. Threats: Res. Pract. 1 (3) (2020) 1–23, http://dx.doi.org/10.1145/3399668.

[101] W. Melicher, C. Fung, L. Bauer, L. Jia, Towards a lightweight, hybrid approach for detecting dom xss vulnerabilities with machine learning, in: Proceedings of the World Wide Web Conference, WWW, ACM, Ljubljana, Slovenia, 2021, pp. 2684–2695, http://dx.doi.org/10.1145/3442381.3450062.

[102] B. Ayeni, J. Sahalu, K. Adeyanju, Detecting cross-site scripting in web applications using fuzzy inference system, J. Comput. Netw. Commun. 2018 (2018) 1–10, http://dx.doi.org/10.1155/2018/8159548.

[103] D.R. Kuhn, R.N. Kacker, Y. Lei, Introduction to Combinatorial Testing, First ed., Chapman & Hall/CRC, 2013, http://dx.doi.org/10.5555/2500930.

[104] D. Simos, B. Garn, J. Zivanovic, M. Leithner, Practical combinatorial testing for xss detection using locally optimized attack models, in: Proceedings of the 2019 IEEE 12th International Conference on Software Testing, Verification and Validation Workshops, ICSTW, IEEE, Xi'an, China, 2019, pp. 122–130, http://dx.doi.org/10.1109/ICSTW.2019.00040.

[105] J. Bozic, B. Garn, I. Kapsalis, D. Simos, S. Winkler, F. Wotawa, Attack pattern-based combinatorial testing with constraints for web security testing, in: 2015 IEEE International Conference on Software Quality, Reliability and Security, QRS, IEEE, Vancouver, BC, Canada, 2015, pp. 207–212, http://dx.doi.org/10.1109/QRS.2015.38.

[106] D. Simos, K. Kleine, L. Ghandehari, B. Garn, Y. Lei, A combinatorial approach to analyzing cross-site scripting (xss) vulnerabilities in web application security testing, in: Proceedings of the International Conference on Testing Software and Systems, ICSS, Springer, Graz, Austria, 2016, pp. 70–85, http://dx.doi.org/10.1007/978-3-319-47443-4_5.

[107] B. Garn, M. Radavelli, A. Gargantini, M. Leithner, D. Simos, A fault-driven combinatorial process for model evolution in xss vulnerability detection, in: Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE, Springer, Graz, Austria, 2019, pp. 207–215, http://dx.doi.org/10.1007/978-3-030-22999-3_19.

[108] B. Garn, D.S. Lang, M. Leithner, D.R. Kuhn, R. Kacker, D.E. Simos, Combinatorially xssing web application firewalls, in: 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW, IEEE, Porto de Galinhas, Brazil, 2021, pp. 85–94, http://dx.doi.org/10.1109/ICSTW52544.2021.00026.

[109] M.A. Ahmed, F. Ali, Multiple-path testing for cross site scripting using genetic algorithms, J. Syst. Archit. 64 (2016) 50–62, http://dx.doi.org/10.1016/j.sysarc.2015.11.001.

[110] A.W. Marashdih, Z.F. Zaaba, H.K. Omer, Web security: Detection of cross site scripting in php web application using genetic algorithm, Int. J. Adv. Comput. Sci. Appl. 8 (5) (2017) 64–75, http://dx.doi.org/10.14569/IJACSA.2017.080509.

[111] A. Avancini, M. Ceccato, Security testing of web applications: A search-based approach for cross-site scripting vulnerabilities, in: Proceedings of the 11th International Working Conference on Source Code Analysis and Manipulation, SCAM, IEEE, Williamsburg, VA, USA, 2011, pp. 85–94, http://dx.doi.org/10.1109/SCAM.2011.7.

[112] J.C. King, Symbolic execution and program testing, Commun. ACM 19 (7) (1976) 385–394, http://dx.doi.org/10.1145/360248.360252.

[113] Z. Liu, Y. Fang, C. Huang, Y. Xu, GAXSS: Effective payload generation method to detect XSS vulnerabilities based on genetic algorithm, Secur. Commun. Netw. (2022) 2031924, http://dx.doi.org/10.1155/2022/2031924.

[114] F. Caturano, G. Perrone, S. Romano, Discovering reflected cross-site scripting vulnerabilities using a multiobjective reinforcement learning environment, Comput. Secur. 103 (2021) 102204, http://dx.doi.org/10.1016/j.cose.2021.102204.

[115] C.J.C.H. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (3) (1992) 279–292, http://dx.doi.org/10.1007/BF00992698.

[116] Y. Frempong, Y. Snyder, E. Al-Hossami, M. Sridhar, S. Shaikh, Hijax: Human intent javascript xss generator, in: Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT, SciTePress, Milano, Italy, 2021, pp. 798–805, http://dx.doi.org/10.5220/0010583807980805.

[117] M. Foley, S. Maffeis, HAXSS: Hierarchical reinforcement learning for XSS payload generation, in: Proceedings of IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom, Wuhan, China, 2022, pp. 147–158, http://dx.doi.org/10.1109/TrustCom56396.2022.00031.

[118] S. Lee, S. Wi, S, S. Son, Link: Black-box detection of cross-site scripting vulnerabilities using reinforcement learning, in: Proceedings of the ACM Web Conference, WWW'22, Lyon, France, 2022, pp. 743–754, http://dx.doi.org/10.1145/3485447.3512234.

[119] S. Lekies, B. Stock, M. Johns, 25 Million flows later - large-scale detection of dom-based xss, in: Proceedings of the ACM Conference on Computer and Communications Security, CCS, ACM, Berlin, Germany, 2013, pp. 1193–1204, http://dx.doi.org/10.1145/2508859.2516703.

[120] S. Bensalim, D. Klein, T. Barber, M. Johns, Talking about my generation: Targeted dom-based xss exploit generation using dynamic data flow analysis, in: Proceedings of the 14th European Workshop on Systems, EuroSec, ACM, Edinburgh, Scotland, UK, 2021, pp. 27–33, http://dx.doi.org/10.1145/3447852.3458718.

[121] R. Wang, G. Xu, X. Zeng, X. Li, Z. Feng, Tt-xss: A novel taint tracking based dynamic detection framework for dom cross-site scripting, J. Parallel Distrib. Comput. 118 (2018) 100–106, http://dx.doi.org/10.1016/j.jpdc.2017.07.006.

[122] A. Kiezun, P. Guo, K. Jayaraman, M. Ernst, Automatic creation of sql injection and cross-site scripting attacks, in: Proceedings of the International Conference on Software Engineering, ICSE, IEEE, Vancouver, BC, Canada, 2009, pp. 199–209, http://dx.doi.org/10.1109/ICSE.2009.5070521.

[123] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, M.D. Ernst, Finding bugs in dynamic web applications, in: Proceedings of the 2008 International Symposium on Software Testing and Analysis, ISSTA, ACM, Seattle, WA, USA, 2008, pp. 261–272, http://dx.doi.org/10.1145/1390630.1390662.

[124] J. Pan, X. Mao, Domxssmicro: A micro benchmark for evaluating dom-based cross-site scripting detection, in: Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom, IEEE, Tianjin, China, 2016, pp. 208–215, http://dx.doi.org/10.1109/TrustCom.2016.0065.

[125] J. Pan, X. Mao, Detecting dom-sourced cross-site scripting in browser extensions, in: Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME, IEEE, Shanghai, China, 2017, pp. 24–34, http://dx.doi.org/10.1109/ICSME.2017.11.

[126] K. DeVoe, Innovations affecting us – what's greasemonkey, and do i want it in the library? Against Grain 20 (3) (2008) 12, http://dx.doi.org/10.7771/2380-176X.2425.

[127] S. Van Acker, N. Nikiforakis, L. Desmet, W. Joosen, F. Piessens, Flashover: Automated discovery of cross-site scripting vulnerabilities in rich internet applications, in: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, ASIACCS, ACM, Seoul, Korea, 2012, pp. 12–13, http://dx.doi.org/10.1145/2414456.2414462.

[128] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, G. Vigna, Saner: Composing static and dynamic analysis to validate sanitization in web applications, in: Proceedings of the 2008 IEEE Symposium on Security and Privacy, SP, IEEE, Oakland, CA, USA, 2008, pp. 387–401, http://dx.doi.org/10.1109/SP.2008.22.

[129] M.T. Louw, V. Venkatakrishnan, Blueprint: Robust prevention of cross-site scripting attacks for existing browsers, in: Proceedings of the 30th IEEE Symposium on Security and Privacy, SP, IEEE, Oakland, CA, USA, 2009, pp. 331–346, http://dx.doi.org/10.1109/SP.2009.33.

[130] G. Iha, H. Doi, An implementation of the binding mechanism in the web browser for preventing xss attacks: Introducing the bind-value headers, in: Proceedings of the International Conference on Availability, Reliability and Security, ARES, IEEE, Fukuoka, Japan, 2009, pp. 966–971, http://dx.doi.org/10.1109/ARES.2009.19.

[131] I. Parameshwaran, E. Budianto, S. Shinde, H. Dang, A. Sadhu, P. Saxena, Auto-patching dom-based xss at scale, in: Proceedings of the 2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE, ACM, Bergamo, Italy, 2015, pp. 272–283, http://dx.doi.org/10.1145/2786805.2786821.

[132] T. Scholte, W. Robertson, D. Balzarotti, E. Kirda, Preventing input validation vulnerabilities in web applications through automated type analysis, in: Proceedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference, COMPSAC, IEEE, Izmir, Turkey, 2012, pp. 233–243, http://dx.doi.org/10.1109/COMPSAC.2012.34.

[133] L. Shar, H. Tan, Automated removal of cross site scripting vulnerabilities in web applications, Inf. Softw. Technol. 54 (5) (2012) 467–478, http://dx.doi.org/10.1016/j.infsof.2011.12.006.

[134] R. Wang, X. Jia, Q. Li, S. Zhang, Machine learning based cross-site scripting detection in online social network, in: Proceedings of the 16th IEEE International Conference on High Performance Computing and Communications, HPCC, IEEE, Paris, France, 2014, pp. 823–826, http://dx.doi.org/10.1109/HPCC.2014.137.

[135] S. Rathore, P. Sharma, J. Park, Xssclassifier: An efficient xss attack detection approach based on machine learning classifier on snss, J. Inform. Process. Syst. 13 (4) (2017) 1014–1028, http://dx.doi.org/10.3745/JIPS.03.0079.

[136] F. Mereani, J. Howe, Preventing cross-site scripting attacks by combining classifiers, in: Proceedings of the 10th International Joint Conference on Computational Intelligence, IJCCI, SciTePress, Seville, Spain, 2018, pp. 135–143, http://dx.doi.org/10.5220/0006894901350143.

[137] J. Zhang, Y.-T. Jou, X. Li, Cross-site scripting (xss) detection integrating evidences in multiple stages, in: Proceedings of the Annual Hawaii International Conference on System Sciences, HICSS, AIS Electronic Library, Grand Wailea, Maui, Hawaii, USA, 2019, pp. 7166–7175, http://dx.doi.org/10.24251/HICSS.2019.860.

[138] X. Li, W. Ma, Z. Zhou, C. Xu, Xss attack detection model based on semi-supervised learning algorithm with weighted neighbor purity, in: Proceedings of the 19th International Conference on Ad-Hoc Networks and Wireless, ADHOC-now, Springer, Bari, Italy, 2020, pp. 198–213, http://dx.doi.org/10.1007/978-3-030-61746-2_15.

[139] Y. Zhou, P. Wang, An ensemble learning approach for xss attack detection with domain knowledge and threat intelligence, Comput. Secur. 82 (2019) 261–269, http://dx.doi.org/10.1016/j.cose.2018.12.016.

[140] Y. Fang, C. Huang, Y. Xu, Y. Li, Rlxss: Optimizing xss detection model to defend against adversarial attacks based on reinforcement learning, Future Internet 11 (8) (2019) 1–13, http://dx.doi.org/10.3390/fi11080177.

[141] X. Zhang, Y. Zhou, S. Pei, J. Zhuge, J. Chen, Adversarial examples detection for xss attacks based on generative adversarial networks, IEEE Access 8 (2020) 10989–10996, http://dx.doi.org/10.1109/ACCESS.2020.2965184.

[142] Q. Wang, H. Yang, G. Wu, K.-K.R. Choo, Z. Zhang, G. Miao, Y. Ren, Black-box adversarial attacks on xss attack detection model, Comput. Secur. 113 (2022) 102554, http://dx.doi.org/10.1016/j.cose.2021.102554.

[143] L. Chen, C. Tang, J. He, H. Zhao, X. Lan, T. Li, XSS adversarial example attacks based on deep reinforcement learning, Comput. Secur. 120 (2022) 102831, http://dx.doi.org/10.1016/j.cose.2022.102831.

[144] F. Mokbal, D. Wang, X. Wang, L. Fu, Data augmentation-based conditional wasserstein generative adversarial network-gradient penalty for xss attack detection system, PeerJ Comput. Sci. 6 (2020) 1–20, http://dx.doi.org/10.7717/peerj-cs.328.

[145] M. Johns, B. Engelmann, J. Posegga, Xssds: Server-side detection of cross-site scripting attacks, in: Proceedings of the Annual Computer Security Applications Conference, ACSAC, IEEE, Anaheim, CA, USA, 2008, pp. 335–344, http://dx.doi.org/10.1109/ACSAC.2008.36.

[146] F. Sun, L. Xu, Z. Su, Client-side detection of xss worms by monitoring payload propagation, in: Proceedings of the 2009 European Symposium on Research in Computer Security, ESORICS, Springer, Saint-Malo, France, 2009, pp. 539–554, http://dx.doi.org/10.1007/978-3-642-04444-1_33.

[147] S. Sundareswaran, A. Squicciarini, Xss-dec: A hybrid solution to mitigate cross-site scripting attacks, in: Proceedings of the 26th Annual IFIP Annual Conference on Data and Applications Security and Privacy, DBSec, Springer, Paris, France, 2012, pp. 223–238, http://dx.doi.org/10.1007/978-3-642-31540-4_17.

[148] D. Das, U. Sharma, D. Bhattacharyya, Detection of cross-site scripting attack under multiple scenarios, Comput. J. 58 (4) (2013) 808–822, http://dx.doi.org/10.1093/comjnl/bxt133.

[149] A. Nunan, E. Souto, E. Dos Santos, E. Feitosa, Automatic classification of cross-site scripting in web pages using document-based and url-based features, in: Proceedings of the IEEE Symposium on Computers and Communications, ISCC, IEEE, Cappadocia, Turkey, 2012, pp. 702–707, http://dx.doi.org/10.1109/ISCC.2012.6249380.

[150] S. Goswami, N. Hoque, D. Bhattacharyya, J. Kalita, An unsupervised method for detection of xss attack, Int. J. Netw. Secur. 19 (5) (2017) 761–775, http://dx.doi.org/10.6633/IJNS.201709.19(5).14.

[151] F. Mokbal, W. Dan, A. Imran, L. Jiuchuan, F. Akhtar, W. Xiaoxi, Mlpxss: An integrated xss-based attack detection scheme in web applications using multilayer perceptron technique, IEEE Access 7 (2019) 100567–100580, http://dx.doi.org/10.1109/ACCESS.2019.2927417.

[152] F. Mereani, J. Howe, Rule extraction from neural networks and other classifiers applied to xss detection, in: Proceedings of the International Joint Conference on Computational Intelligence, IJCCI, Springer, Vienna, Austria, 2021, pp. 359–386, http://dx.doi.org/10.1007/978-3-030-70594-7_15.

[153] F.M.M. Mokbal, D. Wang, X. Wang, Detect cross-site scripting attacks using average word embedding and support vector machine, Int. J. Netw. Secur. 4 (1) (2022) 20–28, http://dx.doi.org/10.6633/IJNS.20220124(1).03.

[154] P. Nagarjun, S. Ahamad, Ensemble methods to detect xss attacks, Int. J. Adv. Comput. Sci. Appl. 11 (5) (2020) 695–700, http://dx.doi.org/10.14569/IJACSA.2020.0110585.

[155] V. Malviya, S. Rai, A. Gupta, Development of web browser prototype with embedded classification capability for mitigating cross-site scripting attacks, Appl. Soft Comput. 102 (2021) 106873, http://dx.doi.org/10.1016/j.asoc.2020.106873.

[156] F. Mokbal, W. Dan, W. Xiaoxi, Z. Wenbin, F. Lihua, Xgbxss: An extreme gradient boosting detection framework for cross-site scripting attacks based on hybrid feature selection approach and parameters optimization, J. Inform. Secur. Appl. 58 (2021) 102813, http://dx.doi.org/10.1016/j.jisa.2021.102813.

[157] R. Kadhim, M. Gaata, A hybrid of cnn and lstm methods for securing web application against cross-site scripting attack, Indones. J. Electr. Eng. Comput. Sci. 21 (2) (2020) 1022–1029, http://dx.doi.org/10.11591/ijeecs.v21.i2.pp1022-1029.

[158] Y. Fang, Y. Xu, P. Jia, C. Huang, Providing email privacy by preventing webmail from loading malicious xss payloads, Appl. Sci. (Switzerland) 10 (13) (2020) 1–17, http://dx.doi.org/10.3390/app10134425.

[159] P. Chaudhary, B.B. Gupta, X. Chang, N. Nedjah, K.T. Chui, Enhancing big data security through integrating xss scanner into fog nodes for smes gain, Technol. Forecast. Soc. Change 168 (2021) 120754, http://dx.doi.org/10.1016/j.techfore.2021.120754.

[160] Z. Liu, Y. Fang, C. Huang, J. Han, Graphxss: An efficient xss payload detection approach based on graph convolutional network, Comput. Secur. (2022) 102597, http://dx.doi.org/10.1016/j.cose.2021.102597.

[161] H. Pan, Y. Fang, C. Huang, W. Guo, X. Wan, GCNXSS: An attack detection approach for cross-site scripting based on graph convolutional networks, KSII Trans. Internet Inform. Syst. 16 (12) (2022) 4008–4023, http://dx.doi.org/10.3837/tiis.2022.12.013.

[162] P. Wurzinger, C. Platzer, C. Ludl, E. Kirda, C. Kruegel, Swap: Mitigating xss attacks using a reverse proxy, in: Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems, SESS 2009, IWSESS, IEEE, Vancouver, BC, Canada, 2009, pp. 33–39, http://dx.doi.org/10.1109/IWSESS.2009.5068456.

[163] D. Mitropoulos, K. Stroggylos, D. Spinellis, A. Keromytis, How to train your browser: Preventing xss attacks using contextual script fingerprints, ACM Trans. Privacy Secur. 19 (1) (2016) 1–31, http://dx.doi.org/10.1145/2939374.

[164] S. Gupta, B. Gupta, Robust injection point-based framework for modern applications against xss vulnerabilities in online social networks, Int. J. Inform. Comput. Secur. 10 (2–3) (2018) 170–200, http://dx.doi.org/10.1504/IJICS.2018.091455.

[165] C.C.C.K.T. Chaudhary Pooja, B.B. Gupta, Xsspro: Xss attack detection proxy to defend social networking platforms, in: Proceedings of the 9th International Conference on Computational Data and Social Networks, CSoNet, Springer, Dallas, TX, USA, 2020, pp. 411–422, http://dx.doi.org/10.1007/978-3-030-66046-8_34.

[166] J. Pazos, J.-S. Légaré, I. Beschastnikh, Xsnare: Application-specific client-side cross-site scripting protection, in: Proceedings of the 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER, IEEE, Honolulu, HI, USA, 2021, pp. 154–165, http://dx.doi.org/10.1109/SANER50967.2021.00023.

[167] M. Gebre, K.-S. Lhee, M. Hong, A robust defense against content-sniffing xss attacks, in: Proceeding of the 6th International Conference on Digital Content, Multimedia Technology and Its Applications, IDC, IEEE, Seoul, South Korea, 2010, pp. 315–320.

[168] A. Javed, J. Schwenk, Towards elimination of cross-site scripting on mobile versions of web applications, in: Proceedings of the 14th International Workshop on Information Security Applications, WISA, Springer, Jeju Island, Korea, 2014, pp. 103–123, http://dx.doi.org/10.1007/978-3-319-05149-9_7.

[169] J. Shanmugam, M. Ponnavaikko, A solution to block cross site scripting vulnerabilities based on service oriented architecture, in: Proceedings of the 6th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2007; 1st IEEE/ACIS International Workshop on E-Activity, IWEA, IEEE, Melbourne, VIC, Australia, 2007, pp. 861–866, http://dx.doi.org/10.1109/ICIS.2007.45.

[170] J. Shanmugam, M. Ponnavaikko, Behavior-based anomaly detection on the server side to reduce the effectiveness of cross site scripting vulnerabilities, in: Proceedings of the 3rd International Conference on Semantics, Knowledge, and Grid, SKG, IEEE, Xi'an, China, 2007, pp. 350–353, http://dx.doi.org/10.1109/SKG.2007.63.

[171] E. Kirda, N. Jovanovic, C. Kruegel, G. Vigna, Client-side cross-site scripting protection, Comput. Secur. 28 (7) (2009) 592–604, http://dx.doi.org/10.1016/j.cose.2009.04.008.

[172] T. Jim, N. Swamy, M. Hicks, Defeating script injection attacks with browser-enforced embedded policies, in: Proceedings of the 16th International Conference on World Wide Web, WWW, ACM, Banff, Alberta, Canada, 2007, pp. 601–610, http://dx.doi.org/10.1145/1242572.1242654.

[173] M. Johns, Script-templates for the content security policy, J. Inform. Secur. Appl. 19 (3) (2014) 209–223, http://dx.doi.org/10.1016/j.jisa.2014.03.007.

[174] S. Stamm, B. Sterne, G. Markham, Reining in the web with content security policy, in: Proceedings of the 19th International Conference on World Wide Web, WWW, ACM, Raleigh, North Carolina, USA, 2010, pp. 921–930, http://dx.doi.org/10.1145/1772690.1772784.

[175] A. Doupé, W. Cui, M. Jakubowski, M. Peinado, C. Kruegel, G. Vigna, Dedacota: Toward preventing server-side xss via automatic code and data separation, in: Proceedings of the ACM Conference on Computer and Communications Security, CCS, ACM, Berlin, Germany, 2013, pp. 1205–1216, http://dx.doi.org/10.1145/2508859.2516708.

[176] M. Fazzini, P. Saxena, A. Orso, Autocsp: Automatically retrofitting csp to web applications, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, ICSE, IEEE, Florence, Italy, 2015, pp. 336–346, http://dx.doi.org/10.1109/ICSE.2015.53.

[177] X. Pan, Y. Cao, S. Liu, Y. Zhou, Y. Chen, T. Zhou, Cspautogen: Black-box enforcement of content security policy upon real-world websites, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS, ACM, Vienna, Austria, 2016, pp. 653–665, http://dx.doi.org/10.1145/2976749.2978384.

[178] J. Iqbal, R. Kaur, N. Stakhanova, Polidom: Mitigation of dom-xss by detection and prevention of unauthorized dom tampering, in: Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES, ACM, Canterbury, CA, UK, 2019, pp. 1–10, http://dx.doi.org/10.1145/3339252.3339257.

[179] R. Mui, P. Frankl, Preventing web application injections with complementary character coding, in: Proceedings of the European Symposium on Research in Computer Security, ESORICS, Springer, Leuven, Belgium, 2011, pp. 80–99, http://dx.doi.org/10.1007/978-3-642-23822-2_5.

[180] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, G. Vigna, Cross site scripting prevention with dynamic data tainting and static analysis, in: Proceedings of the Network and Distributed System Security Symposium, NDSS, The Internet Society, San Diego, California, USA, 2007, pp. 1–12.

[181] B. Stock, S. Lekies, T. Mueller, P. Spiegel, M. Johns, Precise client-side protection against dom-based cross-site scripting, in: Proceedings of the 23rd USENIX Security Symposium, USENIX, USENIX, San Francisco, CA, USA, 2014, pp. 655–670.

[182] A. Barth, J. Caballero, D. Song, Secure content sniffing for web browsers, or how to stop papers from reviewing themselves, in: 2009 30th IEEE Symposium on Security and Privacy, SP, IEEE, Oakland, CA, USA, 2009, pp. 360–371, http://dx.doi.org/10.1109/SP.2009.3.

[183] A. Barua, H. Shahriar, M. Zulkernine, Server side detection of content sniffing attacks, in: Proceedings of the 22nd International Symposium on Software Reliability Engineering, ISSRE, IEEE, Hiroshima, Japan, 2011, pp. 20–29, http://dx.doi.org/10.1109/ISSRE.2011.27.

[184] B. Potteiger, Z. Zhang, L. Cheng, X. Koutsoukos, A tutorial on moving target defense approaches within automotive cyber–physical systems, Front. Future Transp. 2 (2022) 1–13, http://dx.doi.org/10.3389/ffutr.2021.792573.

[185] Y. Nadji, P. Saxena, D. Song, Document structure integrity: A robust basis for cross-site scripting defense, in: Proceedings of the 16th Annual Network and Distributed System Security Symposium, 20 of NDSS, The Internet Society, San Diego, California, USA, 2009, pp. 1–20.

[186] E. Athanasopoulos, V. Pappas, A. Krithinakis, S. Ligouras, E.P. Markatos, T. Karagiannis, Xjs: Practical xss prevention for web application development, in: Proceedings of the 2010 USENIX Conference on Web Application Development, WebApps, USENIX, Boston, MA, USA, 2010, pp. 1–13.

[187] H. Shahriar, M. Zulkernine, S2xs2: A server side approach to automatically detect xss attacks, in: Proceedings of the IEEE 9th International Conference on Dependable, Autonomic and Secure Computing, DASC, IEEE, Sydney, NSW, Australia, 2011, pp. 7–14, http://dx.doi.org/10.1109/DASC.2011.26.

[188] S. Gupta, B. Gupta, Smart xss attack surveillance system for osn in virtualized intelligence network of nodes of fog computing, Int. J. Web Serv. Res. 14 (4) (2017) 1–32, http://dx.doi.org/10.4018/IJWSR.2017100101.

[189] S. Gupta, B.B. Gupta, P. Chaudhary, Designing a xss defensive framework for web servers deployed in the existing smart city infrastructure, J. Organizat. End User Comput. 32 (4) (2020) 85–111, http://dx.doi.org/10.4018/JOEUC.2020100105.

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

[191] A. Niakanlahiji, J. Jafarian, Webmtd: Defeating cross-site scripting attacks using moving target defense, Secur. Commun. Netw. 2019 (2019) 1–13, http://dx.doi.org/10.1155/2019/2156906.

[192] Y. Cao, V. Yegneswaran, P.A. Porras, Y. Chen, Pathcutter: Severing the self-propagation path of xss javascript worms in social web networks, in: Proceedings of the Network and Distributed System Security Symposium, NDSS, The Internet Society, San Diego, California, USA, 2012, pp. 1–14.

[193] V.V.N. Bisht Prithvi, Xss-guard: Precise dynamic prevention of cross-site scripting attacks, in: Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, DIMVA, Springer, Paris, France, 2008, pp. 23–43, http://dx.doi.org/10.1007/978-3-540-70542-0_2.

[194] D. Bates, A. Barth, C. Jackson, Regular expressions considered harmful in client-side xss filters, in: Proceedings of the 19th International Conference on World Wide Web, WWW, ACM, Raleigh, North Carolina, USA, 2010, pp. 91–99, http://dx.doi.org/10.1145/1772690.1772701.

[195] R. Pelizzi, R. Sekar, Protection, usability and improvements in reflected xss filters, in: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, ASIACCS, ACM, Seoul, South Korea, 2012, pp. 1–11, http://dx.doi.org/10.1145/2414456.2414458.

[196] S. Gupta, B. Gupta, Xss-immune: A Google chrome extension-based xss defensive framework for contemporary platforms of web applications, Secur. Commun. Netw. 9 (17) (2016) 3966–3986, http://dx.doi.org/10.1002/sec.1579.

[197] S. Lalia, A. Sarah, Xss attack detection approach based on scripts features analysis, in: Proceedings of the 2018 World Conference on Information Systems and Technologies, WorldCIST, Springer, Naples, Italy, 2018, pp. 197–207, http://dx.doi.org/10.1007/978-3-319-77712-2_19.

[198] M. Krishnan, Y. Lim, S. Perumalc, G. Palanisamy, Detection and defending the XSS attack using novel hybrid stacking ensemble learning-based DNN approach, Digit. Commun. Netw. (2022) http://dx.doi.org/10.1016/j.dcan.2022.09.024, in Press.

[199] A.W. Marashdih, Z.F. Zaaba, K. Suwais, N.A. Mohd, Web application security: An investigation on static analysis with other algorithms to detect cross site scripting, Procedia Comput. Sci. 161 (2019) 1173–1181, http://dx.doi.org/10.1016/j.procs.2019.11.230.

[200] A. Hannousse, S. Yahiouche, Handling webshell attacks: A systematic mapping and survey, Comput. Secur. 108 (2021) 102366, http://dx.doi.org/10.1016/j.cose.2021.102366.

[201] A. Hannousse, S. Yahiouche, Towards benchmark datasets for machine learning based website phishing detection: An experimental study, Eng. Appl. Artif. Intell. 104 (2021) 104347, http://dx.doi.org/10.1016/j.engappai.2021.104347.

[202] B. Mihaljević, C. Bielza, P. Larrañaga, Bayesian networks for interpretable machine learning and optimization, Neurocomputing 456 (2021) 648–665, http://dx.doi.org/10.1016/j.neucom.2021.01.138.

[203] A. Ampatzoglou, S. Bibi, P. Avgeriou, M. Verbeek, A. Chatzigeorgiou, Identifying, categorizing and mitigating threats to validity in software engineering secondary studies, Inf. Softw. Technol. 106 (2019) 201–230, http://dx.doi.org/10.1016/j.infsof.2018.10.006.

[204] Y. Zheng, Z. Li, X. Xu, Q. Zhao, Dynamic defenses in cyber security: Techniques, methods and challenges, Digital Communications and Networks 8 (2022) 422–435, http://dx.doi.org/10.1016/j.dcan.2021.07.006.