Journal Pre-proofs

Machine and Deep Learning-based XSS Detection Approaches: A Systematic Literature Review

Isam Kareem Thajeel Thajeel, Khairulmizam Samsudin, Shaiful Jahari Hashim, Fazirulhisyam Hashim

PII: S1319-1578(23)00182-9
DOI: https://doi.org/10.1016/j.jksuci.2023.101628
Reference: JKSUCI 101628

To appear in: *Journal of King Saud University - Computer and Information Sciences*

Received Date: 28 April 2023
Revised Date: 29 May 2023
Accepted Date: 14 June 2023

Please cite this article as: Kareem Thajeel Thajeel, I., Samsudin, K., Jahari Hashim, S., Hashim, F., Machine and Deep Learning-based XSS Detection Approaches: A Systematic Literature Review, *Journal of King Saud University - Computer and Information Sciences* (2023), doi: https://doi.org/10.1016/j.jksuci.2023.101628

Machine and Deep Learning-based XSS Detection Approaches: A Systematic Literature Review

Machine and Deep Learning-based XSS Detection Approaches: A Systematic Literature Review

Isam Kareem Thajeel Thajeel, Khairulmizam Samsudin, Shaiful Jahari Hashim, Fazirulhisyam Hashim

Department of Computer and Communication Systems Engineering, Faculty of Engineering, Universiti Putra Malaysia (UPM), Serdang 43400, Selangor, Malaysia

Authors' contributions

Isam Kareem Thajeel Thajeel: Conceptualization, Methodology, Investigation, Writing-original draft.

Khairulmizam Samsudin: Conceptualization, Investigation, Supervision, Writing – review & editing.

Shaiful Jahari Hashim: Writing – review & editing, Supervision.

Fazirulhisyam Hashim: Writing – review & editing, Supervision.

All authors reviewed and edited the manuscript. All authors read and approved the final manuscript.

Corresponding authors:

1. **Khairulmizam Samsudin**
   khairulmizam@upm.edu.my

2. **Isam Kareem Thajeel Thajeel**
   isamkthajeel@gmail.com

Isam Kareem Thajeel Thajeel received the B. Eng. degree in software engineering from Baghdad, Iraq, in 2010, the M.Sc. degree in electrical and computer engineering from Altinbas university, Istanbul, Turkey, in 2017, and he is currently a Ph.D. student in Computer and embedded system engineering at Computer & Communication Systems Engineering Department of Universiti Putra Malaysia. His current research interests include cybersecurity, web application, applying the artificial intelligence and machine learning techniques.

Khairulmizam Samsudin is currently a senior lecturer at Computer & Communication Systems Engineering Department of Universiti Putra Malaysia. He received his Ph.D. from the University of Glasgow, UK in 2006 and received his B. Eng. degree in Computer and Communications from Universiti Putra Malaysia in 2002. His research interests include device modelling, embedded systems, robotics, security, web services and distributed computing.

Shaiful Jahari Hashim received the B.Eng. degree in electrical and electronics engineering from the University of Birmingham, U.K., in 1998, the M.Sc. degree from the National University of Malaysia, in 2003, and the Ph.D. degree from Cardiff University, U.K., in 2011. He is currently an Associate Professor with the Department of Computer and Communication Systems Engineering, Faculty of Engineering, Universiti Putra Malaysia (UPM). His research interests include cloud computing, the Internet of Things (IoT), network security, and non-linear wireless measurement systems.

Fazirulhisyam Hashim received the B.Eng. degree from Universiti Putra Malaysia, in 2002, the M.Sc. degree from Universiti Sains Malaysia, in 2006, and the Ph.D. degree from The University of Sydney, Australia, in 2010. He is currently an Associate Professor with the Department of Computer and Communication Systems Engineering, Universiti Putra Malaysia. He also leads the Communication Network Laboratory, and is a member of the Wireless and Photonics Networks (WiPNET) Research Group. His research interests include wireless communication networks include mobile networks, network and computer security, wireless sensor networks, software-defined networking, network function virtualization, blockchain, and vehicular communication.

Abstract

Web applications are paramount tools for facilitating services providing in the modern world. Unfortunately, the tremendous growth in the web application usage has resulted in a rise in cyberattacks. Cross-site scripting (XSS) is one of the most frequent cyber security attack vectors that threaten the end

user as well as the service provider with the same degree of severity. Recently, an obvious increase of the Machine learning and deep learning ML/DL techniques adoption in XSS attack detection. The goal of this review is to come with a special attention and highlight of Machine learning and deep learning approaches. Thus, in this paper, we present a review of recent advances applied in ML/DL for XSS attack detection and classification. The existing proposed ML/DL approaches for XSS attack detection are analyzed and taxonomized comprehensively in terms of domain areas, data preprocessing, feature extraction, feature selection, dimensionality reduction, Data imbalance, performance metrics, datasets, and data types. Our analysis reveals that the way of how the XSS data is preprocessed considerably impacts the performance and the attack detection models. Proposing a full preprocessing cycle reveals how various ML/DL approaches for XSS attacks detection take advantage of different input data preprocessing techniques. The most used ML/DL and preprocessing stages have also been identified. The limitations of existing ML/DL-based XSS attack detection mechanisms are highlighted to identify the potential gaps and future trends.

Keywords: cross-site scripting (XSS) attacks, web application security, cybersecurity, Machine learning, Deep learning.

1 Introduction

Web applications have become an essential mean for businesses and government organizations to streamline operations, enhance service quality, and reach a wider audience through the internet. Online services offer significant benefits to users as well (X. Wang et al., 2022). However, such advantages are also combined with risks, as web applications contain user information that are frequent targets for cyber-attacks, exposing confidential data belonging to both the organization and its users to potential threats.

Cross-Site Scripting (XSS) attacks are defined as one of the most frequent cyber security vulnerabilities in web applications (Sarmah et al., 2020; Tariq et al., 2021; Zhou & Wang, 2019). XSS attacks are mainly caused due to the lack of suitable sanitization of user inputs. Such a vulnerability is exploited by the attackers through injecting malicious scripts into legitimate web pages that have this vulnerability to lure victims to visit them (Sarmah et al., 2018). This leads to the execution of the malicious script in the victim's browser, enabling the attacker to control and attack the victim's browser. XSS attacks can be used to reveal sensitive information, such as usernames, login credentials, and financial data. They can also be used to launch high-level attacks, including Distributed Denial of Service (DDOS) attacks, Cross-Site Request Forgery (CSRF) attacks, Remote Command/Code Execution (RCE) attacks, and more (Mokbal et al., 2021). According to the Open Web Application Security Project (OWASP), XSS has ranked 4th, 4th, 1st, 3rd, 7th, and 3rd among the top 10 global web application security risks since 2004, 2007, 2010, 2013, 2017, and in 2021 respectively (*OWASP Top Ten Web Application Security Risks | OWASP*). XSS is still a major threat to user security and privacy, as evidenced by several high-profile attacks in recent years, which resulted in serious information leakage and immeasurable economic loss.

Recent research has shown that artificial intelligence (AI) techniques, such as machine learning (e.g., decision tree (DT), random forest (RF), support vector machine (SVM), etc.) and Deep learning (e.g., convolutional neural network (CNN), recurrent neural network (RNN), long short-term memory (LSTM), etc.) (Kaur et al., 2023). ML/DL-based techniques can improve the detection of XSS attacks and overcome the issues that obstacle the traditional methods of XSS attack detection, including input validation, static analysis, and dynamic analysis (M. Liu et al., 2019; R. Wang et al., 2018). The main advantage of ML/DL-based techniques is the ability to learn from data and adapt to new or unknown forms of XSS attack.

It has been observed that the majority of researchers have addressed the problem of XSS attack detection with ML/DL-based approaches from the perspective of data generation, feature extraction, selection, reduction, and classification. Fig. 1 illustrates the XSS attack scenario and detection system using ML/DL techniques. These techniques are based on different principles, and how to effectively exploit their advantages to address XSS attack detection tasks in different domains remains an open research issue. For example, due to the high dimensionality and complexity of the XSS data, a common solution is to use data preprocessing techniques, which reduce the dimensionality and thus enable the researchers to deal with these high-dimensional spaces. Preprocessing methods can affect detection performance and should be carefully considered in the design of XSS detection methods.

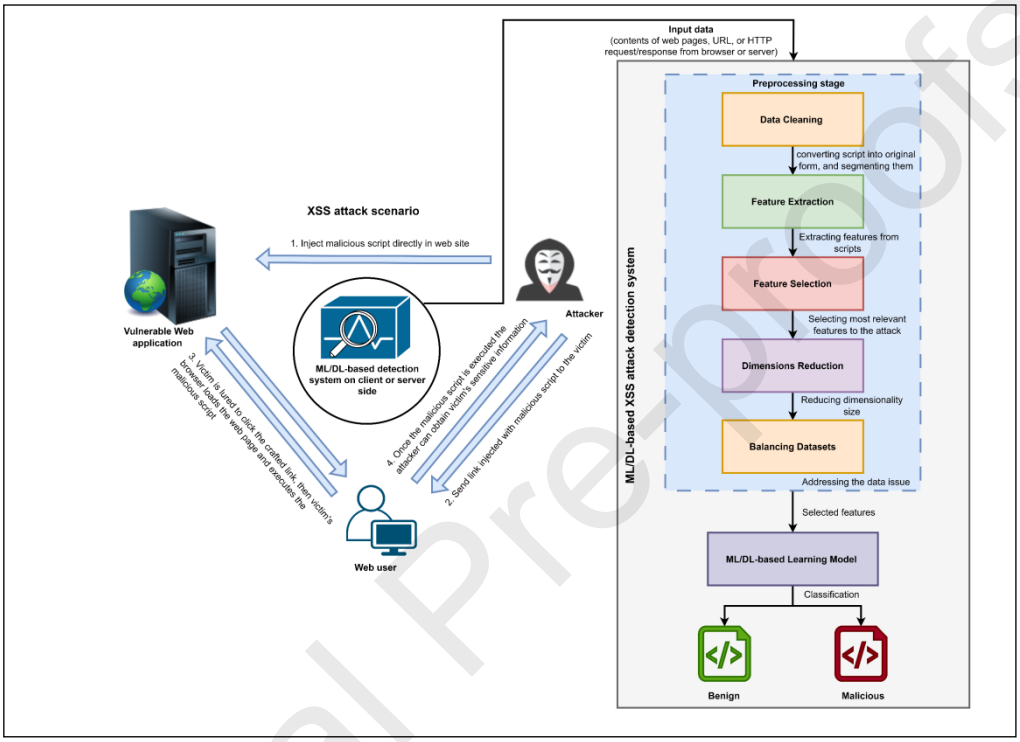

Fig. 1. Illustration of the XSS attack scenario and detection system using ML/DL approaches.

To address the above-given issue, a systematic and comprehensive literature review is required and contributes to develop the community. Thus, we came up with a methodology of systematic literature review (SLR) to examine comprehensively and classify the current state of ML/DL-based XSS detection techniques. Moreover, an overview of the preprocessing steps of input XSS data is presented to show the critical influence of the XSS attack detection performance and the effectiveness of ML/DL-based models. This review will also emphasize the limitations of the existing ML/DL-based methods. By providing this overview, this study aims to support future research in the field of XSS attack detection using ML/DL techniques. Additionally, this review will facilitate researchers in this domain by providing a clear understanding of the current state of ML/DL-based XSS detection techniques, and helping them to identify areas for future research and improvement. Our contributions in this article are:

- It provides a comprehensive insight into the recent ML/DL-based XSS detection mechanisms and extracts deep details. We provide a comprehensive overview of 52 recent ML/DL-based methods for XSS attack detection.

- Review the current state of the art of ML/DL-based XSS detection approaches, including domain areas, preprocessing methods, feature extraction techniques, feature selection methods, datasets, learning methods, performance evaluation criteria, and feature types.
- Analyzing the recent trend of adversarial XSS attack examples that threaten ML/DL-based techniques.
- The analysis findings shed light on the limitations of the current ML/DL-based approaches and potential future directions.

This paper is organized as follows. Section 2 summarizes the existing literature review work related to XSS attack detection, and Section 3 presents the methodology of our conducted SLR. Our research analysis and findings are introduced in Section 4. The limitations of the current ML/DL-based approaches and future directions are presented in Section 5.

2 Related Work

In this section, an overview of previous review studies that cover XSS attack detection is provided; also, the most related works and a comparison of this review with them are presented.

The work by (John-Otumu et al., 2018) briefly reviews only the traditional techniques for detecting and preventing XSS attacks, focusing on an architectural framework and solution location. It found that most solutions are focused on the client end and lack the ability to self-learn and detect new XSS attacks.

The first Systematic Literature Review (SLR) that stands alone in the domain of XSS attack detection was presented by (Hydara et al., 2015). This review focuses on publications from 2000 to 2012. It explores the proposed traditional techniques and solutions for addressing XSS, including techniques, tools, and methods. The study also examines the type of XSS attack addressed and the proposed solution, whether it is focused on preventing XSS attacks or vulnerabilities, detecting them, or removing vulnerabilities from programs.

In the review of (P. M. D. Nagarjun & Ahamad, 2020), 171 research papers on cross-site scripting published between 2002 and 2019 are analyzed. The finding states that the majority of existing XSS prevention methods are Dynamic analysis, Static analysis, Proxy based method, Filter based method, and machine learning. The review categorizes existing methods based on their implementation location, such as client-side and server-side solutions. However, this study discussed the general methodology of the solutions presented in the studies.

In the study of (Stency & Mohanasundaram, 2021), a review of various XSS attack detection methods using various performance metrics was presented. The research focuses on finding an efficient tool and detection method for XSS in various web applications by reviewing numerous works published between 2019 and 2020. The research is based on XSS detection and prevention mechanisms, specifically those based on traditional methods, deep learning, and artificial intelligence techniques. The article compares various methods and presents the limitations of the work.

In the review of (S. Gupta & Gupta, 2017), an analysis of the major concerns for web applications and Internet-based services is presented by referencing the Website Security Statistics Report of White Hat Security. The article highlights some of the serious vulnerabilities in modern web applications, specifically focusing on the XSS attack as the top vulnerability in today's web applications. The article discusses the taxonomy of XSS attacks and provides a detailed analysis of the exploitation, traditional detection, and prevention tools of XSS attacks.

In the survey of (Rodríguez et al., 2020), 67 researches were reviewed to collect information on tools, AI and traditional methods used by the scientific community to detect and mitigate XSS attacks. The results show a trend towards traditional methods over artificial intelligence techniques. Specifically, it found a trend in proposals that analyze the content of web pages (13.20%) and tools for web browsers (16.98%). Additionally, current studies found a low tendency to use artificial intelligence techniques to detect or mitigate XSS attacks, specifically using web classifiers (9.43%).

The survey of (Marashdih et al., 2019) reviewed the enhanced approach of combining static analysis techniques with other algorithms such as genetic algorithm, pattern matching, and machine learning to improve the detection of XSS vulnerability in PHP web applications.

In the study (Kaur & Garg, 2021), the methodologies used to detect XSS attacks are analyzed, including supervised learning, unsupervised learning, reinforcement learning, deep learning, and metaheuristic algorithms. The survey found that while there is a significant amount of research on supervised and unsupervised machine learning detection models, there is a lack of research on semi-supervised, reinforcement, and deep learning neural networks for XSS detection.

The survey of (M. Liu et al., 2019) discusses the classification of XSS and demonstrates attack processes of common XSS exploitation scenarios through a demo website. It also reviews and compares recent research results on XSS detection, dividing them into three categories: static analysis methods, dynamic analysis methods, and hybrid analysis methods (a combination of static and dynamic analysis, a combination of static analysis or dynamic analysis with ML techniques). The survey classifies 30 studies into these categories, provides an overall comparative analysis, lists their strengths and weaknesses, and highlights the types of XSS vulnerabilities detected by each method.

Table 1

Related studies on XSS attack detection survey.

Authors	Static analysis	Dynamic analysis	Hybrid analysis	AI models	Preprocessing	Feature extraction	Feature Selection	Dimensionality reduction	Performance metrics	Datasets	XSS adversarial examples	No. Of articles
(John-Otumu et al., 2018)	√	√	×	×	×	×	×	×	×	×	×	-
(Hydara et al., 2015)	√	√	√	×	×	×	×	×	×	×	×	115
(Nagarjun & Ahamad, 2020)	√	√	√	√	×	×	×	×	×	×	×	171
(Stency & Mohanasundaram, 2021)	√	√	×	√	×	×	×	×	√	√	×	-
(Gupta & Gupta, 2017)	√	√	√	×	×	×	×	×	×	×	×	65

(Rodríguez et al., 2020)	√	√	√	√	×	×	×	×	×	×	×	67
(Marashdih et al., 2019)	√	×	×	√	×	×	×	×	×	×	×	-
(Kaur & Garg, 2021)	√	√	√	√	×	×	×	×	×	×	×	-
(M. Liu et al., 2019)	√	√	√	√	×	×	×	×	×	×	×	30
Our review	×	×	×	√	√	√	√	√	√	√	√	52

As shown in Table 1., previous review articles on XSS detection can be broadly classified into two categories: those that focus on traditional methods for XSS detection (e.g., static analysis, dynamic analysis, and Hybrid analysis) and those that examine the use of AI methods in conjunction with traditional methods. However, studies that have examined AI-based methods for XSS detection have often lacked a comprehensive analysis of the preprocessing, feature extraction, feature selection, learning methods, dataset types, evaluation, and validation stages employed in these methods. This limitation in previous literature highlights the need for a more in-depth insight into these techniques in the context of ML/DL-based XSS attack detection. Since the efficiency of the AI models is based on the preprocessing methods as well as learning models.

3 Research Methodology

This Systematic Literature Review (SLR) follows the guidelines proposed by (Kitchenham et al., 2015; Kuhrmann et al., 2017). There are three stages to the SLR process. The steps of defining Research questions (RQs), creating, and verifying review protocols will be included in the first phase, which is planning. The second phase covers the discovery and collection of related research, data extraction, as well as the information synthesis procedure, while the third phase covers writing and validating the review.

3.1 Phase 1: Plan Review

This initial step of the SLR research technique specifies the significant analysis/research problems, the design of assessment protocols, and the appropriate searching technique. Thus, four main research questions with several sub-questions are developed based on the purpose of this article which is to have a comprehensive insight into the existing ML/DL-based approaches for XSS attack detection and to identify current gaps and reveal potential future research direction. The proposed research questions are shown in Table 2.

Table 2.

The proposed research questions.

RQ #1	Application areas	a. What are the application areas that the proposed XSS detection approaches implemented? b. Which area is the most covered by researchers?
RQ #2	Preprocessing methods	a. What are the most common preprocessing methods implemented in XSS attack detection? b. What Data Cleaning methods are applied in the XSS attack detection? c. What are the methods used for feature extraction in XSS attack detection? d. What are the most common methods used for feature selection in XSS attack detection? e. What are the most common methods used for dimension reduction in XSS attack detection? f. What are the most common methods used for balancing data in XSS attack detection?
RQ #3	ML/DL-based detection methods	a. What kind of ML/DL-based methods are used in detecting XSS attacks? b. What is the most common ML/DL methods used for XSS detection? c. Are studies considering the adversarial XSS attacks against ML/DL detection models?
RQ #4	Datasets and evaluation metrics	a. What datasets are used in training and testing the proposed XSS attack detection? b. What performance measurements are most used in XSS attack detection evaluation? c. What are the validation steps that may impact the performance of the XSS detection?

3.1.1 Review Protocols

The search for similar publications using specific keywords and scholarly sources was focused during the creation and verification of the review methodology.

3.1.2 Searching Keywords

To ensure that the evaluation is focused on XSS attacks and applicable methodologies for ML/DL-based approaches. We attempted to focus our search on the most relevant terms. So, we initiated with the terms and then moved on to the next steps:

- Taking the main phrases from our main study questions and extracting them.
- Using several distinct spellings of the words
- Integrating keywords from relevant papers into our search terms

- Table 3 shows using the major options to find the most immediately relevant literature and adding “OR operator” and “AND operator”.

Table 3
Review Protocols

ID	Keywords
1	(“Cross site scripting” OR “Cross-site scripting” OR “XSS”) AND (“Attack” OR “Injection” OR “Vulnerability”) AND (“Artificial Intelligence” OR “AI” OR “Machine Learning “OR “ML” OR “Deep Learning “OR “DL”)

3.1.3 Literature Resources

To discover eligible papers for primary review investigations, five online scholarly libraries such as ACM Digital Library[1], Scopus[2], Science Direct[3], Springer[4], and IEEE Xplore[5] are used. These libraries will provide the broadest coverage of high-quality publications on our subject, including ISI and Scopus-indexed articles. Each of these databases' advanced search features was used to build the search phrase. The years 2018 through 2023 were included in our search.

We observed that some papers address web application attacks, JavaScript attacks, or intrusion detection systems; they also have referred to XSS in their abstracts explicitly or implicitly. To obtain the relevant papers, we decided to generalize our search to include such articles. This way, both papers with a main or second focus on XSS are retrieved. To alleviate the impact of such a decision about word search position and to avoid potentially missing relevant studies, we conducted a recursive review backward and forward snowballing on approved papers. Therefore, references and citations of approved papers are screened for relevance regardless of the presence of search strings in the titles or abstract.

3.2 Phase 2: Conduct Review

We conduct the review in this phase based on the research questions, keywords, and protocols. According to the criteria depicted in Table 4 (a and b), this phase focuses primarily on the inclusion and exclusion criteria.

3.2.1 Study Selection

Fig. 2., depicts the entire study selection of our SLR methodology. The obtained articles were subject to two main screening stages; firstly, in the online search, 1050 articles were found. A total of 150 papers were shortlisted after screening the title, keywords, inclusion, and exclusion criteria. Table 4 clarifies the inclusion and exclusion criteria. There were 38 duplicated articles and appeared in multiple

[1] (https://dl.acm.org)
[2] (https://www.scopus.com)
[3] (https://www.sciencedirect.com/)
[4] (https://link.springer.com)
[5] (https://ieeexplore.ieee.org)

databases, and 92 articles related to different domains, were removed. After completing the full reading, 52 articles are selected from the list.

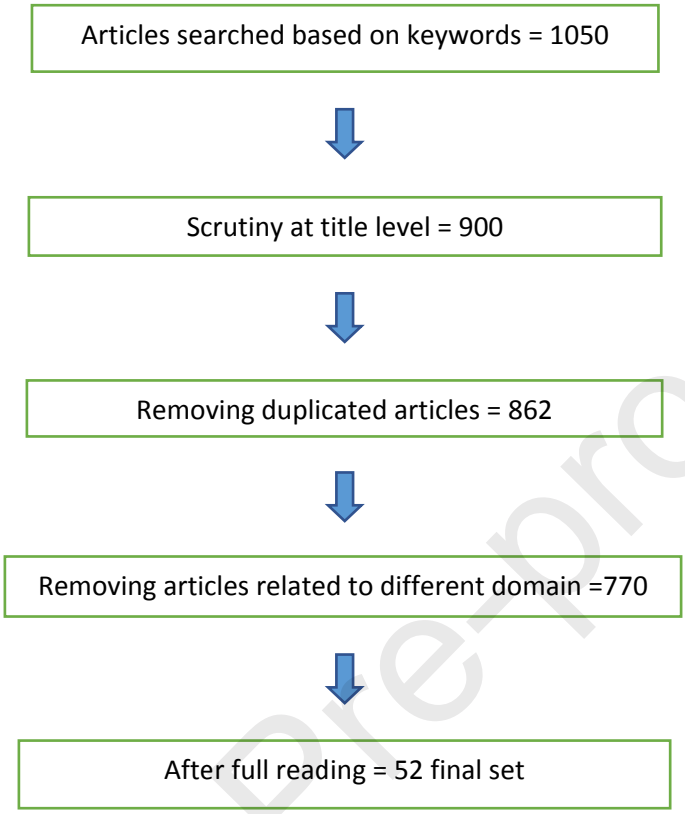

Fig. 2. Study selection Process

Table 4 (a).

Inclusion Criteria

Inclusion Criteria
The research was focused solely on XSS attacks detection
The research was adopted ML/DL techniques for XSS attack detection
The research was proposed for JavaScript attacks detection and considered XSS attacks
The research was related to web attack detection and pay attention of XSS attacks

Only English-published journal and conference articles
Articles are published within a period of 2018-2023

Table 4(b).

Exclusion Criteria

Exclusion Criteria
Studies that were unrelated to XSS attacks and the domain will be skipped
Unsubscribed articles

A set of inclusion and exclusion criteria are considered for selecting the relevant articles, as shown in Table 4: (a & b). The parameters for identifying relevant articles based on keywords are described. Identical articles and those that do not address the research questions are omitted.

To include only the reputable and quality studies, the obtained articles were subject to strict quality assessment steps. To achieve that, 4 quality assessment questions are proposed, as shown in Table 5. The questions applicable to the studies selected are displayed in the quality assessment checklist. The questions are largely meant to collect studies that are more relevant, detailed, and thorough than the other research topics. Thus, only those studies met with quality assessment questions are considered. Thus, it was necessary to carefully read the main parts of the paper, such as the introduction and methodology and results sections, which contained most of the information needed to be examined to answer the proposed research questions and quality assessment.

Table 5.

Quality Checklist

No	**Questions**
1	Have the studies focused on the XSS attacks?
2	Does the study use ML/DL methods?
3	Does the research use a clear solution or adequate methodology?

4	Has the study been published in a well-known Journal or conference?

3.2.2 Data Extraction

To collect the data required to answer our research questions and make contributions. The extracted data is based on what is highlighted in Table 2.

3.2.3 Information Synthesis

The obtained information was gathered manually at this point to solve the research questions. We utilized the narrative synthesis method to answer our research questions. As a result, we will present our findings in the form of tables and charts. Each selected article is thoroughly analyzed by extracting the implicit and explicit detailed information that is required to answer our research questions. Based on the domain information, the selected articles are classified into different application domains (**RQ#1**). For learning models and types, the employed model in each paper is analyzed to extract information such as the type of ML/DL algorithm, classifier, and parameters used in the learning model (**RQ#3**). On the other hand, to analyze the methods of preprocessing data, the articles were classified based on data cleaning methods, feature extraction methods, feature selection methods, dimensionality reduction methods, dimension size, feature types, and data representation types (**RQ#2**). Finally, to shed light on the evaluation and validation process of papers, information such as datasets, sources of the used datasets, data type, evaluation metrics, validation methods and limitations are gathered (**RQ#4**).

4 Analysis of XSS detection literature review

In this section, we summarize and provide our findings. We present a comprehensive analysis and recommendations in the area of XSS attack detection from different aspects, including domain application areas, data preprocessing, learning methods for XSS detection, evaluation metrics, validation method, datasets and data types. The study findings of our comprehensive review are demonstrated in Table 9.

A total of 52 articles are analyzed in this paper, and these articles are published within years between 2018 and 2023. There were 3 studies in 2018, but there is an ascending trend for studies that applied ML/DL in 2019 and 2020, with 8 and 11, respectively. The number of studies in 2021, 2022, and 2023 is 11, 15 and 4, respectively. It is obvious that more than 70 % of selected articles were published between 2020 and 2022; this point demonstrates an upward trend in the number of proposed ML/DL-based solutions employed in the last three years. This is an indicator that the researchers are paying attention to improve the XSS detection by using ML/DL techniques, with the raising of XSS attacks within these years. Fig. 3., depicts the number of studies conducted each year.

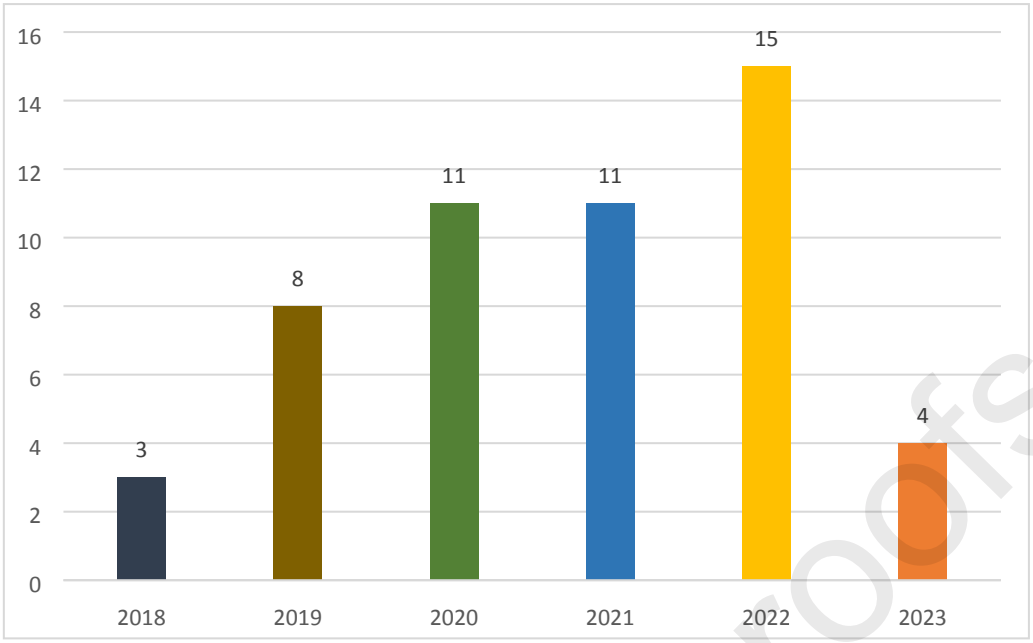

Fig. 3. Distribution of selected studies per year.

4.1 Application Areas

To answer the **RQ#1**, 52 studies are analyzed and categorized to show at which areas the XSS detection are applied. From the analysis, we can classify the studies into six categories: Client-side, Server-side, Hybrid client-server, Internet-of-Things (IoT), Mobile, and others, **RQ#1(a)**. It's worth notifying that our categorization is based mainly on the location of the approaches' implementation and the type of malicious payload scripts used in the evaluation process. Firstly, for the **Client-side** category, some approaches are found in the literature implemented or designed for the client-side. This can be done by configuring the web browser's HTML parser or JavaScript engines. Such developing requirements are considered complicated and costly. Furthermore, we found other methods are suggested to be implemented as an external extension or plug-in as an external tool for the browser. Downloading of external extensions or plug-in tools is not a straightforward step for some users; above that, such external tools are also directly vulnerable to XSS attacks. However, the implementation on client-side can identify and detect all types of XSS through analyzing the user input, URL parameters, and HTTP response most efficiently. We categorize all the adversarial XSS detection and generation methods as client-side methods, since the primary goal of these methods is to generate sophisticated XSS attacks to attack the client side, and they also proposed a detection method for evaluation. Secondly, **Server-side** methods that are proposed and implemented on the application server and have the ability to work with HTTP request or HTTP response messages, more specifically, different HTTP request or response datasets used in the evaluation of the proposed methods. Thirdly, in the **Hybrid client-server** category, the implementation of such methods is based on both the client and server sides to analyze the malicious payloads. Fourthly, **Internet-of-Things (IoT)** methods are proposed or implemented with a simulation for IoT applications such as industrial monitoring and management systems (SCADA) (Abaimov & Bianchi, 2019), content management systems (WordPress) (Lu et al., 2022), Fog nodes networks and clouds (Chaudhary et al., 2021), and Smart Cities application (Kuppa et al., 2022). Fifthly, the **Mobile application** category is the methods implemented and evaluated using a mobile application as datasets (R. Yan et al., 2018). Finally, in the **others category**, the majority of selected articles in this paper are classified as others category because such studies are not specified in which domain their solutions may be applied and used general datasets. Basically, their focus is only on proposing a solution for XSS attack detection regardless of wherever it may be implemented. To answer **RQ #1(b)**, Fig. 4., demonstrates the percentage of studies based on domain categorization in XSS attack detection. More

than 34% of studies are not specified for which domains are proposed, which indicates that the focus on proposing a security detection solution rather than focusing on the domain for implementation. On the other hand, only one study stands alone for XSS attack detection on mobile application. This indicates that the mobile application needs more attention in the future. Through our analysis, we discovered that the few studies in IoT and mobile application domains is due to the lack of public dataset related to IoT and mobile application, or the studies in these domains do not disclose their dataset, which restricts the advancement in XSS attack detection in these domains.

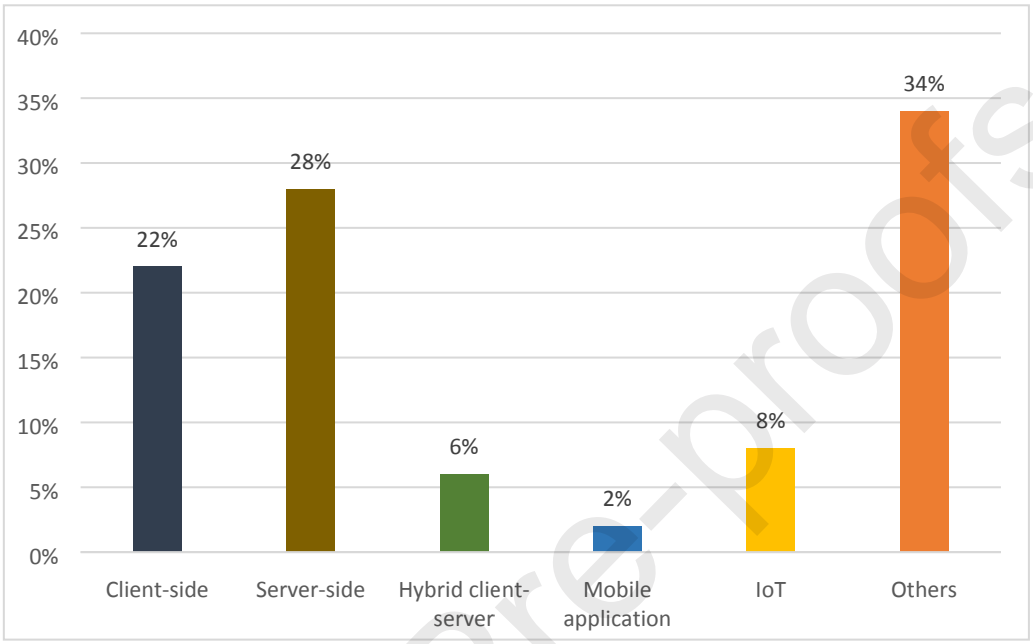

Fig. 4. Percentage of studies by domain categorization in XSS attack detection.

4.2 Preprocessing Methods

Preprocessing can be defined as the process of transforming the data from a complex form into another simple and high-quality form of data that can be understood easily by the machine learning models in the learning process. The importance of the preprocessing step is its ability to preserve and maintain the high performance of the learning model through enabling the analysis model to learn sound knowledge from the data. In addition, the XSS attack datasets are known in text form that needs to be preprocessed and converted to extract features with high dimensions. **RQ #2(a),** The preprocessing consists of several steps in Machine learning, starting from gathering the dataset, cleaning the data, extracting features from the raw data, representing these features in a specific form, following selecting the most relevant features through feature selection methods, then reducing the dimensionality of the features, and at the end and before learning method is applied, the need for handling the imbalance of the classes. However, we found that not all researchers consider all of the preprocessing steps mentioned previously. Some researchers only focused on a specific subset of preprocessing steps most relevant to their dataset and the characteristics of the XSS attacks they are trying to detect. Additionally, other researchers only use a combination of different preprocessing techniques to prepare their dataset for analysis better. It is also worth noting that preprocessing techniques and approaches need to change over time as the attackers are evolving and developing new techniques (Fadel Waheed & Alyasiri, 2023; C. Gupta et al., 2022; Kareem Thajeel et al., 2023; Tariq et al., 2021), so it's important to keep the detection model updated with the preprocessing techniques dynamically overtime whenever the attack evolving is identified.

4.2.1 Data Cleaning

In XSS attack detection, high quality and cleaned data is needed for training and testing the ML/DL-based model to maintain high-performance detection. The data is cleaned in a way to convert the XSS script into the original clean context. Many methods may be applied in this stage to clean the script, such as decoding, removing the null/empty spaces, removing unimportant symbols (e.g., “, “, /, *, <, >, New line, etc.), replacing the URI/IP address, that used by the attacker to escape or bypassing the detection method; this could be the proper answer for **RQ #2(b)**.

4.2.1.1 Removing Noisy data /normalization/Generalization

We found that some researchers use the term normalization and generalization to describe the process of removing the noise or unimportant piece of data. The noise data is included by the attacker intentionally to confuse the detection models and eventually caused them to classify such vectors as benign scripts. An example of noise data, the attacker may split the attack vector into several lines by using a newline symbol, using many other disturbing symbols, including closing tags >, starting tags <, **, /, “, ‘, uppercasing the letters, using numbers, many URI links or IP addresses etc., all of this noise data caused to bypass the detection system and conduct the attack once it is executed by the browsers’ parsers. It is worth noting that this step can be applied after the decoding process. This is because the attacker may insert noisy data into the payload before the encoding process, making it harder to detect.

4.2.1.2 Decoding

Encoding is the most common method used by attackers to create evolved or mutated XSS attacks. This mechanism provides the attacker the ability to evade the traditional detection systems such as Web Application Firewall (WAF) since the browser decodes and escapes such scripts automatically to be parsed; thus, this feature is exploited by the attacker to hide and conduct evolved XSS attacks effectively. Therefore, the decoding process allows for more accurate detection of XSS attacks as the payload is transformed into its original context form. The literature analysis showed that various common decoding methods were implemented including URL decoding, HTML entity decoding, Unicode decoding, Base64 decoding, JSFuck decoding, and Deobfuscation. The HTML Entity, URL and Unicode can be decoded easily by decoding tools since their decoding principle is mapping. Base64 is an encoding method that encrypts information, and the encrypted data is completely unreadable. Since this encryption is reversible, we can use the Base64 decryption tool to decode. JSfuck encoding converts the string into a long text of only 6 symbols([]()!+), which can still be directly executed by the browser. The text after JSfuck encoding is like garbled code, so we must use the open-source tool Jsunfuckit to decode. In addition, Deobfuscation can be done by applying various deobfuscation techniques, such as de-minifying the code or using Deobfuscation. Table 6. shows examples of the XSS payloads after and before the decoding process (Fang et al., 2020). Finally, we found some studies that apply the parsing process by using parsers as a decoding process instead of using specific decoding methods, as an example of these parsers, BeautifulSoup library, html5lib parser, and Esprima parser.

Table 6.

After and before applying the decoding methods.

Decoding method	**XSS payload before decoding**	**XSS payload before decoding**
URL decoding	“%3Cscript%3Ealert(%27XSS%27)%3C%2Fscript%3E"	“<script>alert('XSS')</script>"

Base64 decoding	"PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4="	"<script>alert('XSS')</script>"
Unicode	"\u003Cscript\u003Ealert(\u0027XSS\u0027)\u003C\u002Fscript\u003E"	"<script>alert('XSS')</script>"
JSFuck	"[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+ (![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[] +!+[]+!+[]]+(!![]+[])[+!+[]]][([][(![]+[])[+[]]+ ([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+ (!![]+[])[+[]]+(!..... (show a part)"	alert(4)
Deobfuscation	"eval(function(p,a,c,k,e,d){e=function(c){return c};if(!''.replace(/^/,String)){while(c--){d[c]=k[c]||c}k=[function(e){return d[e]}];e=function(){return'\w+';c=1};while(c--){if(k[c]){p=p.replace(new RegExp('\b'+e(c)+'\b','g'),k[c])}}return p}('0 1=2.3('4');1.5=6;',7,7,'var|x|document|getElementById||xss||innerHTML|XSS'.split('|'),0,{}))"	"var x = document.getElementById ('xss'); x.innerHTML = 'XSS';"

4.2.1.3 Segmentation/Tokenization

Reviewing the literature, we found a variation in the usage of the terms of segmentation and tokenization. Some researchers use the term segmentation, and others use the term Tokenization to refer to the same process of breaking down a text into smaller units or tokens such as words or phrases. However, we will use the term Tokenization in this article to avoid misinterpretation. In XSS detection, tokenization is used to break down the payloads into smaller units for analysis when working with payloads with no spacing. This is because payloads with no spacing can be difficult to analyze as a whole and tokenizing them allows for a more detailed analysis of the individual words or phrases that make up the payload. The researchers used tokenization as a preprocessing step, where the payloads are analyzed to identify the sequences of the payload that assist in malicious XSS attack detection. This can make it easier to apply other preprocessing steps, such as input sanitization and data normalization. Tokenization can also be used as a feature extraction step, where the payloads may be tokenized after preprocessing, and the tokens are used as features for machine learning models. This allows the model to analyze the individual words or phrases in the payloads rather than the payload as a whole.

Furthermore, we found that the researchers used different tokenization methods, such as word-level, character-level, or n-gram tokenization. Also, tokenization can be applied in different ways depending on the specific dataset and the characteristics of the XSS attacks. Furthermore, the segmentation process is performed by Using the NLTK module (Nature Language Toolkit) in the Python library, and the regular expression tokenizer was used to divide the source code into multiple segments based on its semantic structure. We treated each segmented code as a word, as shown in the examples in Table 7.

Tokenization is also used with Abstract Syntax Tree (AST) or JSON files after parsing the malicious JavaScript files. AST is a tree representation of the structure of a JavaScript program, and tokenization is used to break down the AST into smaller units for analysis. Some researchers refer to each node of the AST as a token, while others refer to each line of the AST as a token. Tokenization with AST representation adds an additional layer of analysis that can help to improve the detection of XSS attacks. It allows researchers to analyze the structure and sequence of the payloads, in addition to the individual words or characters.

Table 7.

Examples of tokenization methods and steps.

Tokenization method	Before tokenization	After tokenization	Advantages
Word-level tokenization	"<script>alert("XSS");</script>"	["<script>","alert","(","""","XSS","""",")",";","</script>"]	Word-level tokenization could be useful for identifying the key words that are used in the payload
Character-level tokenization	"<script>alert("XSS");</script>"	["<","s","c","r","i","p","t",">","a","l","e","r","t","(","""","X","S","S","""",")",";","<","/","s","c","r","i","p","t",">"]	character-level tokenization could be useful for identifying encoded or obfuscated characters.
N-gram tokenization	"<script>alert("XSS");</script>"	["<s","sc","cr","ri","ip","pt","t>",">a","al","le","er","rt","t(","(","""","X","XS","SS","S","","")",");",";<","</","/s","sc","cr","ri","ip","pt","t>"]	n-gram tokenization could be useful for detecting patterns in payloads or identify the unknown words and solve the problem of Out-Of-Vocabulary (OOV)

In general, from a data cleaning perspective, **RQ#2(b)**, it has found that the 16% of selected studies apply the three cleaning processes (Normalization, decoding and tokenization) (Fang et al., 2018, 2020; Hu et al., 2023; Kuppa et al., 2022; Lei et al., 2020; Z. Liu et al., 2021; H. Pan et al., 2022; H. Yan et al., 2022), and other 34% of studies do not apply any process for data cleaning (Huang et al., 2021; Kareem Thajeel et al., 2023; V. Malviya et al., 2018; Mohammadi & Namadchian, 2020; Mokbal et al., 2020; Y. Pan et al., 2019; Panigrahi et al., 2022; Rozi et al., 2022; Shahid et al., 2022; Tama et al., 2020; Tariq et al., 2021). Also, 6% of analyzed studies use only the normalization method as a cleaning step (Melicher et al., 2021; Stiawan et al., 2023; Tekerek, 2021), while 8% of the studies combined the normalization with tokenization method for cleaning the XSS data (Ghaleb et al., 2022; Z. Liu et al., 2023; W. Yang et al., 2019; G. Zhang et al., 2019), 10% of researches combined the normalization with the decoding process for cleaning and converting the data into an original form (Chaudhary et al., 2021; C. Gupta et al., 2022; Li et al., 2020; Lu et al., 2022; Zhou & Wang, 2019). Moreover, 4% of studies cleaned the XSS data with the decoding process only (Abaimov & Bianchi, 2019; V. K. Malviya et al., 2021), while 8% use parsers as an alternative process for decoding the XSS samples (Alazab et al., 2022; Mokbal et al., 2019, 2021; R. Yan et al., 2018). Finally, 14% of studies use only the tokenization method when they clean their training dataset (Akaishi et al., 2019; Z. Cheng et al., 2020, 2021; Maurel et al., 2022; P. Nagarjun & Ahamad, 2020; Niu & Li, 2020; J. Yang et al., 2020). Fig. 5., illustrates the percentage of using the cleaning data method in the analyzed studies.

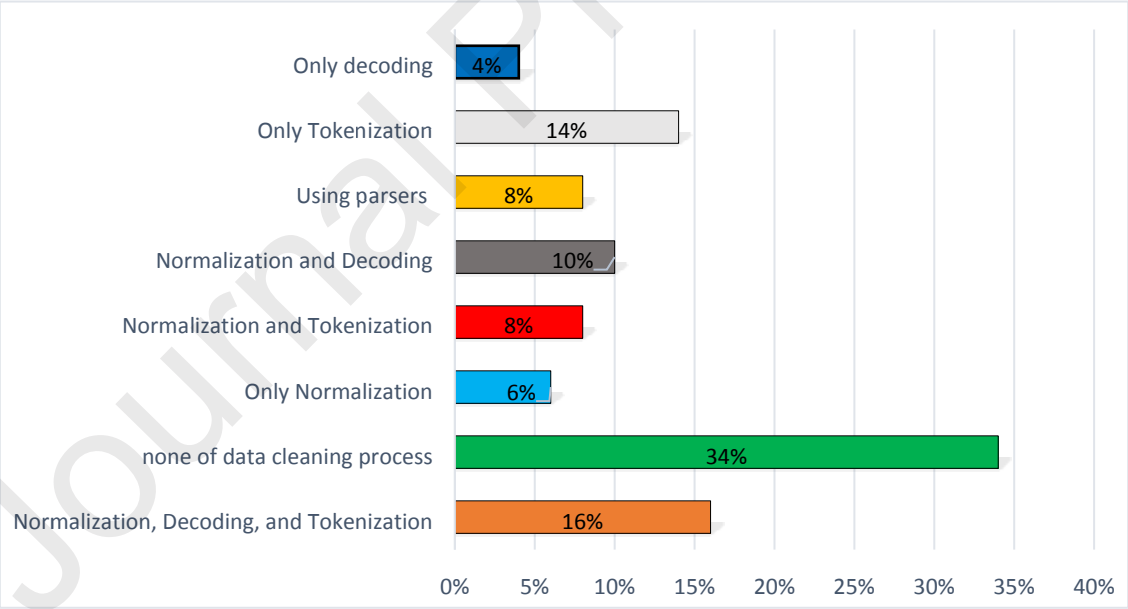

Fig. 5. Percentage of studies applied data cleaning methods.

4.2.2 Feature extraction

Feature extraction process can be defined as the initial and major step in building the learning/prediction model, which plays a significant role in facilitating model learning, especially in the security domain. This process aimed to derive information from the XSS attack scripts, such as attack payloads, HTTP

requests from the user, URL parameters, HTTP response from the server, or JavaScript files, and then represent this information as features in the dataset. Thus, to detect such attacks efficiently, the accurate features need to be extracted and fed to the model for learning.

Generally, there are two common methods used for feature extraction are static analysis and dynamic analysis. Indeed, these methods are traditional methods used for XSS attacks detection (Mereani & Howe, 2018; Rodríguez et al., 2020). The static analysis method involves analyzing the cleaned data to extract the features without executing the script code. On the other hand, Dynamic analysis focuses on analyzing the behavior and data flow of the script during the running time. However, these two analysis methods are subjected to various limitations. Static analysis is often prone to noise data and false positives as it cannot understand the logic and structure in the code lines. Static analysis is also vulnerable to obfuscation and encoding techniques that are employed by the attacker to hide the malicious code. Dynamic analysis consumes a lot of time and resources since it needs to run a large amount of benign code (M. Liu et al., 2019; R. Wang et al., 2018). Thus, the researchers leveraged these methods by using the ML/DL techniques. Static and dynamic analysis are used as prefilter methods with ML/DL techniques where generally adopted in the preprocessing stage and, more specifically, within the feature extraction process.

Through reviewing the state-of-the-art of ML/DL-based XSS attack detection approaches, we found that almost all selected articles adopt the static analysis mechanism with ML/DL approaches for feature extraction. Thus, we can classify these approaches into Domain-expertise-based, statistical-based, NLP-based, contextual analysis, convolutional-based, and graph-based methods, **RQ#2(C)**. In this case, the features are extracted directly from XSS attack data, which can be done manually or automatically.

4.2.2.1 Domain expertise-based feature extraction

In the context of XSS attack detection, some researchers extract features directly from the AST or preprocessed XSS payload vectors through using the knowledge or domain expertise and these features also are called handcrafted features. This is achieved by pre-defining some specific characteristics, e.g., length of the payload, if the encoding is applied or not, the frequency appearance time, presence of certain functions, attributes, event holders, tags, etc., as an indicator of XSS attack. Characteristics are defined based on the manual observation analysis of the XSS data. Many cheat sheets[6][7] are available provide details of XSS attacks, such as JS functions, Tags, html attributes, event handlers, methods, and CSS.

Such a method was adopted to extract features from JS files, as the study of (Alazab et al., 2022) extracted 170 features for learning the ML models. Moreover, using the same method, 70 features were extracted from the traffic flow (HTTP request) and XSS payloads in the work of (Li et al., 2020). Another domain expertise-based approach proposed for feature extraction was found in the study of (Cui et al., 2018), which extracts 21 features from HTTP requests to describe the behavior of HTTP requests. In the work of (Niu & Li, 2020), eight features were extracted with sound classification effects to augment the original data (Niu & Li, 2020). (Zhou & Wang, 2019) and (Tariq et al., 2021) extracted 30 handcrafted features directly from the XSS payload. Although these handcrafted features have achieved a good performance; however, these selected features that perform well in one training dataset may perform poorly in other XSS datasets.

4.2.2.2 Statistical-based feature extraction

Based on the features predefined by the Domain expertise-based feature extraction method and using statistical mechanisms, the features can be extracted and represented into more informative and useful

[6] https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html
[7] https://portswigger.net/web-security/cross-site-scripting/cheat-sheet

representations to be used in machine learning or deep learning models. To achieve the high performance of machine learning, such XSS features need to be processed and transformed into a representation form that the machine learning can learn from them to build the prediction model.

By observing the selected studies, we found some researchers have used a statistical-based feature extraction method to locate key elements of malicious requests using three mechanisms for feature representation with statistical form.

- **Numeration (0 or 1)**: Such a method depends on the appearance of a certain feature or keyword within the XSS sample. The binary representation is given to the features such as 1 if the features are existing in the attack vector, else is given to features that appear in the payload attack. The numeration mechanism is used in the works of (Tariq et al., 2021) that extract 30 handcrafted features and represent them in a binary form. The study of (C. Gupta et al., 2022) adopts this method with a dimension size of 500. Also, (Lu et al., 2022) employed the numeration mechanism to extract 37 statistical features. However, we found that this step is done manually, which leads to consuming time, and may miss extracting some crucial features.

- **Counting (0, 1, 2, …, M)**: This method is based on the frequency of features appearing in the given XSS samples in the dataset. Some researchers count the characters' length of the payloads, the length of the URL, the appearance times of specific HTML tags, HTML events, HTML attributes, special characters, script tags, URL cookies, JS functions, etc. The counting mechanism was used by the study of (Mokbal et al., 2019) to extract initially 41 handcrafted features, and then the same authors extended their extraction method to extract 77 features; such feature extraction method also was improved in the recent work of (Mokbal et al., 2021) to extract a total feature size of 167. Furthermore, other researchers (V. Malviya et al., 2018) performed the counting approach with statistical-based feature extraction to extract the JS and web page features with a total of 34 features. The feature extraction method in previous work, was extended in the work of (V. K. Malviya et al., 2021) to extract 43 features (10 web pages features and 33 JS script features). The frequency of the feature's appearance is not always can be used as a feature of XSS attack indicator, since the attacker is able to avoid such feature by creating an XSS attack vector with the same size of benign vector samples.

- **Encoding (type, value) or (code, type):** This method transforms the original data into an encoded pattern (numeric tokens) by mapping each command/symbol into a pair of values, where one value serves as a simple semantic label for each feature, for example, the researchers (Abaimov & Bianchi, 2019) use 0 for operators, 1 for expressions, and 2 for escape symbols and the other value is the numeric token for the word or symbol. This allows the neural network to understand the role of each symbol and reduces the training needs. The encoded data is processed as a pattern by the neural network and padded to match a fixed-width sequence, as queries or lines of code have variable lengths. Such a method was implemented in the works of (Abaimov & Bianchi, 2019; Chaudhary et al., 2021; P. Nagarjun & Ahamad, 2020), as shown in Fig. 6.

```
<img src=1 href=1 onerror="javascript:
    alert(1)"></img>
Remove noise: < img src = href =
    onerror = " javascript : alert ( ) "
     > < / img >
Encode: (1,0) (4,1) (4,1) (4,1) (5,0)
    (4,1) (2,1) (10,0) (3,1) (2,1) (3,1)
     (3,1) (6,1)
Merge: [1,0,4,1,4,1,4,1,5,0,4,1,
    2,1,10,0,3,1,2,1,3,1,3,1,6,1]
```

Fig. 6. Encoding mechanism for Feature extraction.

4.2.2.3 NLP-based feature extraction (vectorization)

Due to the large amount of handcrafted, statistical features needed to be extracted and the performance of different ML/DL-based XSS detection methods, it is hard to use such methods in real scenarios detection tasks. Thus, Natural Language Processing (NLP) NLP-based feature extraction method is adopted by authors to improve the performance of ML/DL techniques within XSS attack detection. NLP is a branch of artificial intelligence created to deal with natural language and improve the interaction between humans and machines. The NLP technique refers to how the words are arranged (syntax) and the conveyed meaning of the text (semantic).

Reviewing the literature, we found that word embedding is the most NLP-based feature extraction method adopted. **Word embedding** is a collection of NLP models that represent words in the form of low dimensional vectors called embedding, with the purpose of improving the performance of machine-learning-networks in textual analysis of the dataset (Ge & Moh, 2017; Xu et al., 2016). Word embedding is a technique that converts text data into a numerical representation that can be used by a machine learning model. By using word embedding, the researcher transforms the raw text data into a numerical representation that captures the semantic meaning of the text. If words have the same meaning on a text document, they will have the exact representation. In general, the unique words in the text data will be extracted and one-hot encoded in the preprocessing process for easier computation. However, in extensive text data, computing thousands of one-hot vectors could be inefficient as most values in one-hot vectors are 0 and yield results that have mostly 0 values (Phung & Mimura, 2021).

Word vector is a preprocessing method employed to learn word embedding and vectorize a word from documents as a vector (Word2vec) (Mikolov et al., 2013). This vector describes the feature of its corresponding word if two words have the same meaning; their feature vector converted from the word vector model will be similar to each other. Two main models of Word vector (word2vec) are Skim-gram and Continuous bag-of-words (CBoW), which consist of two shallow layers of neural networks trained to reconstruct the linguistic context of the words. It is observed that the Word2vec model is used in studies of (Kuppa et al., 2022; Mohammadi & Namadchian, 2020) as a method for feature extraction and representation.

The word vector model has been extended to include models such as Paragraph vector (Doc2Vec). Such a model uses a document's fixed-length feature vector to represent the document across a collection of input documents (Le & Mikolov, 2014). This model addresses the two primary flaws of the bag-of-words model, which are their inability to handle longer texts and their disregard for the semantics of individual words. The Doc2Vec framework works by assigning a distinct ID vector to each paragraph in matrix D as well as to every word in matrix W. Dov2Vec contains two primary strategies that are directly influenced by the Skip-gram and CBoW models. The Doc2Vec was employed in the works of (Ngoc & Mimura, 2021; Phung & Mimura, 2021) as a feature extraction and representation method for malicious JavaScript attack detection.

In the study of (Maurel et al., 2022), a comparison was introduced of two-word embedding methods Word2vec as an NLP model and Code2Vec (Alon et al., 2019) as Programming Language Processing (PLP) model. Code2Vec suggests a representation of code derived from data in the AST. In particular, the method starts with generating an AST for a specific block of program code. Then, Code2Vec suggests logging every feasible route between AST leaves. A "path context" is a set of interconnected pathways. The three-part path context identifies the starting point (source), the ending point (target), and the route used to get there (which is the sequence of nodes of the tree). Finally, their results show that PLP representations, which encode more knowledge about the semantics of the language into the model, are better suited than NLP representations. However, Code2Vec implementations only support C# and Java source codes. Their focus on determining the type of code and its efficiency is not approved yet with the attack detection in the security domain.

Other researchers (Akaishi et al., 2019; Niu & Li, 2020) combine the word2vec model with previous statistical-based feature extraction methods to extract the most frequent sensitive keywords in a vector representation to be used with deep learning approaches. In the work of (Ghaleb et al., 2022), the features are extracted at the word level and character level using N-gram model and statistical-based text representation, Term Frequency-Inverse Document Frequency (TF-IDF) (Tahmasebi & Risse, 2011), each one of them is applied with a different dataset. In another study by (Melicher et al., 2021), a CBoW model was used with the domain expertise method to represent the predefined Keywords.

4.2.2.4 Contextual analysis/ semantic analysis

The previous method is still limited to inability to extract the deep logical features of complex semantics (Zheng & Yin, 2022). Obfuscated data significantly boosts the detection complexity, and there are several encoding and obfuscation techniques to choose from. Some research approaches have dealt with the attack context as a special natural language to obtain semantic meaning from the context. Thus, several NLP models are adopted by researchers within the XSS attack detection context, as has been explained in sub-section 4.2.2.3. Context scripts do resemble natural languages in certain respects. They share features like being sequences and following certain syntactic constraints. Nevertheless, the script's syntax is more variable, and the dependence connection between statements is not established by their distance; this implies that the semantic information cannot be captured by just traversing the abstract syntax tree from AST or the token sequence from the payload (Song et al., 2020). As a result, using simple natural language processing methods to mine and extract deep semantic information becomes challenging.

To handle such a problem, the study by (Qin et al., 2018) introduced an approach to learn the semantics of malicious segments of payload using a Recurrent Neural Network (RNN) algorithm with an attention mechanism. Nevertheless, the RNN is subjected to problems of the gradient disappearance and gradient explosion problems. (Stiawan et al., 2023). To avoid the problem of RNN and extract the hidden state information, the study of (Stiawan et al., 2023) employed a Long-short Term Memory (LSTM) algorithm. The study by (Yu et al., 2018) provides a technique for modelling HTTP traffic that employs Bi-directional Long Short-Term Memory (Bi-LSTM) with an attention mechanism. The semantics of attacks can be learned by such techniques, but these models still have limitations. When dealing with samples, including HTTP requests/responses, JavaScript files, or URLs, they are disregarded as anything more than a useless generic string made of characters. To deal with this point, other researchers (Fang et al., 2018; Lei et al., 2020) combined the Word2Vec model with the LSTM algorithm to explore the deep semantic features and enhance the feature vectors extracted by the Word2Vec model. However, in NLP, the meaning of a word often depends on the words that come before and after it. Thus, some researchers use a bidirectional LSTM (Bi-LSTM) instead of a regular LSTM algorithm for semantic feature extraction because the Bi-LSTM can capture both forward and backward contexts in a sequence. In contrast, regular LSTM can only capture forward context. Another advantage of Bi-LSTM is that it can capture long-term dependencies in a sequence, which can be difficult for regular LSTMs. This is because Bi-LSTM has two separate hidden states, one for the forward pass and one for the backward pass, which can help it better model complex dependencies in the sequence. (J. Yang et al., 2020) use a single layer of Bi-LSTM with the Word2Vec model. Bi-LSTM was used to generate encoded features and learn the semantics from the word level and segment level. Also, in the study of (Huang et al., 2021), two layers of Bi-LSTM with the Word2Vec model are presented to extract more information from the hidden states. (Fang et al., 2020) also combined the Word2Vec model with a single Bi-directional RNN (Bi-RNN) layer.

4.2.2.5 Convolutional-based features

Recently, deep learning techniques have shown impressive effectiveness in classifying malicious attacks (J. Yang et al., 2020). Hence, the researchers use Convolutional Neural Network layers (CNN) for feature extraction in NLP because the Convolutional layers can effectively capture local patterns and dependencies in text data, such as n-grams or specific word combinations, while still being able to handle variable-length input sequences.

By analyzing the selected studies, we found that Convolutional-based feature extraction methods can be classified into three types based on the combination of feature extraction techniques used. Firstly, Convolutional-based feature extraction with the Word2Vec model: In this approach, Convolutional layers are used to extract features from the Word2Vec embeddings of the input text data. This can be useful for capturing local patterns and dependencies in the text data. Such a method was applied in the studies of (H. Yan et al., 2022; G. Zhang et al., 2019). Secondly, Convolutional-based feature extraction with GRU: In this approach, Convolutional layers are combined with LSTMs or GRUs to extract features from the input text data. This can help capture both local and global patterns in the text data, improving the model's accuracy. This method was proposed in the work of (W. Yang et al., 2019) . Thirdly, Convolutional-based feature extraction with Word2Vec model and Bi-LSTM: In this approach, Convolutional layers are combined with both Word2Vec embeddings and Bi-LSTM networks to extract features from the input text data. This can help capture both local and global patterns in the text data, as well as dependencies in both forward and backward directions. The work of (Hu et al., 2023) suggested a feature extraction method consisting of a combination of Word2Vec, Context analysis (Bi-LSTM), and 3 parallel convolutional layers. (R. Yan et al., 2018) also proposed a hybrid method of the Tri-gram model, Convolutional layers, and a single layer of standard LSTM.

Unlike traditional Convolutional Neural Networks (CNNs), which operate on regular grid-like data structures such as images, some authors use Graph-Convolutional Networks (GCNs) that can operate on arbitrary graphs, which makes them well-suited for NLP tasks that involve analyzing the relationships between words or tokens in a sentence. In the context of feature extraction for NLP tasks, GCNs is used to extract features from graph representations of text, where words or tokens are represented as nodes in the graph, and the relationships between them (such as co-occurrence or syntactic dependencies) are represented as edges. We found that some authors proposed a hybrid approach that combines multiple methods for extracting useful features from XSS data. (H. Pan et al., 2022) (Z. Liu et al., 2021) proposed a method of a combination of Word2Vec (CBOW) and Convolutional-based feature extraction (GCN) for feature extraction. Also (Z. Liu et al., 2023) introduced a method consisting of Word2Vec (CBOW), a single layer of Bi-LSTM, and Convolutional-based feature extraction (GCN).

4.2.2.6 Graph-based features

Based on our literature analysis, we found that a graph-based approach is used as a method for representing complex systems, such as social networks, and encoding node-level attributes as a real-valued matrix. This approach is particularly useful for heterogeneous graphs, where different types of nodes have distinct attribute sets. In the study of (Rozi et al., 2022), the graph-based approach is employed to analyze the structure feature of an AST graph, which represents the overall JS program code, to detect malicious signatures. This method is a state-of-the-art technique that enables the identification of malicious code through its unique characteristics, such as obfuscation techniques implemented with nonstandard tools.

4.2.3 Feature Selection

Feature selection is the step of choosing the most relevant subset features from the entire feature space of the original dataset for using them as input into the ML models. The feature selection process aims to decrease the dimensionality, computational cost, training time of the prediction model, extreme information loss and overfitting, by selecting the optimal relevant features subset and ignoring the redundant or irrelevant features (Chandrashekar & Sahin, 2014). To perform the feature selection process, we found that only 16 of 52 selected articles applied feature selection with XSS attack detection. The feature selection techniques can be classified into five categories as following **RQ #2(e)**:

- **Selection-based knowledge:** In XSS detection, we found that most studies are based on their knowledge (domain expertise) in extracting and selecting features, as explained in Section 4.2.2.1. This can be seen clearly in many works where only a certain number of features are extracted and selected in the study without applying any of the feature selection methods to

measure the importance of each feature. This method has potential limitations such as biased feature selection (may miss the most relevant feature of XSS attacks), a limited feature set that does not capture all of the attack characteristics, lack of generalizability that result in features are specific to a particular dataset or data type, and selecting high feature size can be time-consuming.

- **Filtering method**: Such methods evaluate each feature independently and assign a score or ranking to each feature based on some criteria. The criteria are used to measure the relevance or importance of each feature to the target variable, and features that do not meet the criteria are removed from the dataset. These criteria are correlation coefficient, information Gain, Chi-square, and Document frequency. The correlation coefficient measures the strength and direction of the linear relationship between two variables, and it can be used to filter out features that have a weak correlation with the target variable. Information Gain (IG) measures the reduction in entropy of the target variable when a feature is added to the model and can filter out features that do not provide much information about the target variable. The chi-square test measures the dependence between two categorical variables; thus, features not significantly associated with the target variable are filtered out. Document frequency measures the frequency of a term in a collection of documents and can be used to filter out features that are too rare or too common. (Alazab et al., 2022) introduced a combination method of a correlation coefficient, Information Gain (IG), for selecting relevant features in which 60 relevant features were selected among 170 handcrafted features. (Ghaleb et al., 2022) employed Information Gain (uses entropy to measure impurity). (R. Yan et al., 2018) used information gain, Chi-square test (CHI) and document frequency (DF). Initial space contains 2693 trigram features using the tree feature selection methods; the final selected features by IG, CHI and DF are 835, 835, and 487, respectively. However, Filtering methods can miss out on potentially important interactions between features, as they evaluate each feature independently. Additionally, these methods may not work well for datasets with a large number of features. However, this method ignores the dependencies of features and the interaction between the predictors and feature relevancy selection.
- **Wrapper methods**: These methods evaluate subsets of features by training and testing the model with each subset to determine the most relevant features. This approach is computationally expensive because it requires training and testing the model multiple times with different subsets of features. It requires searching the entire feature space of 2^N, where N denotes the number of features. However, it is often more effective than other feature selection methods because it considers the interaction between features and how they affect the performance of the model. The study of (Mokbal et al., 2021) uses Sequential backward selection (SBS) with IG to select relevant statistical features that selected the 60 most relevant features among 167.
- **Embedded methods:** Some studies use embedded methods for feature selection, which incorporate the feature selection process into the model training. In other words, feature selection is performed as part of the model construction, rather than as a separate pre-processing step. One study by (Panigrahi et al., 2022) uses Multi-Objective Evolutionary Feature Selection (MOEFS), which is an embedded method. In this study, only 5 features were selected from a total of 83 features. The study of (Li et al., 2020) used the CfsSubsetEval feature evaluation method and a greedy search algorithm which selected 14 features among the 70 URL handcrafted features. Another study of (Tariq et al., 2021) uses a Reinforcement learning algorithm to select/modify the relevant pattern based on the classification error rate with statistical features and domain knowledge. However, Embedded methods can be sensitive to the choice of model and hyperparameters. Additionally, they may not always produce interpretable feature selection results.
- **Attention model**: Attention mechanisms are used in some studies as a feature selection method. Attention models focus on the most relevant parts of the input data by assigning weights to each input feature. The studies mentioned in the description use attention mechanisms in conjunction with NLP techniques such as word2vec, LSTM, and Bi-LSTM. The reason for this is that LSTM and Bi-LSTM networks are effective in modeling sequential data, such as language, but

they may not capture long-term dependencies in the data. The attention mechanism can address this problem by allowing the network to selectively focus on the most relevant parts of the input sequence at each time step. Such a method was implemented in the studies of (Fang et al., 2020; Hu et al., 2023; Z. Liu et al., 2023; Maurel et al., 2022; J. Yang et al., 2020). However, attention models may not be effective for feature selection in non-NLP domains.

It is worth to notify a hybrid method that combines the advantages of wrapper and embedded method was introduced by (Kareem Thajeel et al., 2023). The method involves selecting a relevant subset of features for each chunk of streaming data using a Dynamic feature selection-based multi-agent deep Q-network, which is an embedded feature selection method. This approach incorporates the feature selection process into the model training process, allowing for more efficient and effective feature selection. Also, this method involves testing each chunk of data with the selected features from the training of the same chunk. This is similar to the wrapper feature selection method, which evaluates subsets of features by training and testing the model with each subset to determine the most relevant features.

4.2.4 Dimensions Reduction

We found that some studies use dimensionality reduction techniques instead of feature selection in ML/DL-based XSS attack detection methods. The dimensionality reduction methods can help to extract the most relevant information from a large dataset with a high number of features to reduce the risk of overfitting. On the other hand, feature selection methods aim to select a subset of the original features that are most relevant to the problem at hand. In contrast, dimensionality reduction techniques aim to transform the original dataset into a lower-dimensional space while retaining as much relevant information as possible.

By analyzing the ML/DL-based XSS attack detection methods, it is interesting to note that out of the 52 selected studies, only 4 studies used dimensionality reduction techniques, and among these 4 studies, 3 studies used principal component analysis (PCA) and the fourth study used Sparse dimension reduction method, **RQ #2(f)**. This is an indicator that features selection methods are more commonly used in the field of XSS attack detection than dimensionality reduction techniques. There could be several reasons for this observation. First, feature selection methods are generally simpler and faster to implement compared to dimensionality reduction techniques, and they require less computational resources. Second, feature selection methods can be more interpretable and provide insights into which features are most important for the problem at hand. In contrast, dimensionality reduction techniques may be less transparent and harder to interpret. Following are the details of the dimensionality reduction methods with XSS attack detection:

- **Sparse random projection**: Using a sparse random matrix that allows for fast calculation of the data projected, sparse random projection assists in reducing raw data of high dimension to a data of lower dimension. The elements of the random matrix are drawn from a distribution over the numbers -1, 0, and 1 with probabilities of $\frac{1}{2\sqrt{d}}$ for elements 1 and -1, while $1 - (\frac{1}{\sqrt{d}})$ for element 0. This strategy was used in a study of (Kuppa et al., 2022).
- **Principal component Analysis (PCA):** In order to simplify multivariate data, PCA may be used as an unsupervised method. Using the covariance matrix of the data, PCA maps the original data onto a new collection of orthogonal characteristics (principal components). Each original data point may be represented as a linear combination of these main components, the new features, using principal component analysis (PCA). The amount of variation collected by each new characteristic is used to rank the features in terms of importance. The PCA is adopted by the works of (Chaudhary et al., 2021; Y. Pan et al., 2019; Stiawan et al., 2023) for dimensionality reduction.

4.2.5 Balancing the dataset

An uneven distribution of positive and negative samples in a dataset can bias learning in favor of the larger proportion. In the case of a dataset with 95% positive and 5% negative samples, a model trained on this dataset may have high accuracy on positive samples but low accuracy on negative samples. Unbalanced datasets are common in certain fields, such as cybersecurity, where public attacks are relatively rare (Gao et al., 2021). Hence, it's important to evaluate the performance of the model on both positive and negative samples to ensure that the model is not biased towards one class. Regarding web security, web pages are a combination of different languages like HTML and JavaScript, which were previously unstandardized and allow the use of a variety of coding methods that are vulnerable to attacks. The volume of labeled data on XSS attacks with updated instances is limited and extremely unbalanced because of the XSS attacks' heterogeneity and evolved characteristics (Mokbal et al., 2020; Tariq et al., 2021). To address this issue, we found several strategies have been proposed in the XSS domain to improve the performance of ML/DL models. These strategies include **RQ#2(f)**:

- **Using a balanced or semi-balanced dataset:** This involves creating a dataset with an equal or semi-equal number of positive and negative samples to mitigate the bias towards the majority class and improve the model's performance on the minority class. Such method is found in the studies of (Chaudhary et al., 2021; Fang et al., 2020; Huang et al., 2021; Maurel et al., 2022; P. Nagarjun & Ahamad, 2020; H. Yan et al., 2022; W. Yang et al., 2019)**.** Using such a method indicates that the authors aimed to improve the performance of the detection model regardless of the presence of the unbalanced dataset problem. However, this does not fully capture the variability and complexity of the real-world scenarios, the minority class of the positive samples corresponding to XSS attacks.
- **Using the majority of positive samples:** The authors, including (Z. Liu et al., 2023; Tekerek, 2021), used the positive samples in the dataset to be larger than the negative samples to avoid the bias issue. However, by selecting the positive samples as a majority, the authors create a new distribution of positive and negative samples that does not reflect the true prevalence of these classes in the real world. This can impact the generalization performance of the model on unseen data.
- **Using a threshold or weight based on the ratio of samples**: We found studies of (Li et al., 2020; Melicher et al., 2021) involve setting a threshold or weight for the minority class based on the ratio of positive to negative samples. The threshold or weight can be used to adjust the loss function during training so that errors in the minority class are penalized more heavily than those in the majority class. Such a method is limited to more errors in the majority class that can impact the overall performance of the model on both classes.
- **Over-sampling:** This involves increasing the number of samples in the minority class by creating synthetic samples. This method was implemented in the study of (Ngoc & Mimura, 2021). In another study of (Shahid et al., 2022), the number of benign HTTP samples exceeded the malicious counterparts. For this reason, the authors over-sampled minority class data using the Synthetic Minority Over-sampling Technique (SMOTE) (Chawla et al., 2011) to achieve statistical parity between the two class labels. Selected malicious samples were made more similar to one another in the feature space. Further synthetic samples were generated along a line established between these chosen samples in the feature space.

  In the same context, (Mokbal et al., 2020) proposed their own oversampling technique called C-WGAN-GP, which utilizes conditional and Wasserstein GAN with a gradient penalty to address the problem of an imbalanced XSS attack dataset. Unlike traditional approaches, C-WGAN-GP's generative network can produce samples that capture the minority class's overall distribution. To mitigate these issues, a generative adversarial network (GAN) can be employed (Goodfellow et al., 2020). GAN is a deep generative model that can be used for various purposes, such as predictor training, classifier development, and data augmentation. It aims to learn the joint probability distribution of samples and labels from training data (Z. Pan et al., 2019).

- **Under-sampling:** This involves eliminating some of the majority class's samples at random by bringing the minority class's sample size closer to the original. The under-sampling method was adopted in the works of (Panigrahi et al., 2022; Phung & Mimura, 2021). Under-sampling may be harmful because it removes potentially helpful information from the model's training set by excluding data from the majority class (Vluymans, 2019). This can lead to a reduction in the overall performance of the model on both the majority and minority classes.

4.3 ML/DL-based XSS detection methods

The ML/DL approaches that adopted for XSS detection can be classified into two major categories, namely Machine Learning (ML), Deep Learning (DL) **RQ #3(a)**, as summarized in Table 8. To answer **RQ #3(b)**, the ML/DL methods need to be analyzed and reviewed critically, as can be seen in the next subsections and Table 9.

Table 8.

Algorithms of ML/DL most used for XSS attack detection.

Method	Type	Algorithms	No. of articles
ML	Supervised	SVM, DT, NB, KNN, RF, Adaboost, XGBoost, Hidden Markov.	14
	Ensemble learning	Staking, bagging, and boosting ensemble learning methods	5
	Reinforcement learning	RL	5
DL	Supervised	CNN, DNN, RNN, LSTM, Bi-LSTM, Bi-RNN, GRU	24
	Unsupervised	Auto-encoder, GAN	2
Evolutionary algorithms	-	GA	2

4.3.1 Machine learning

Machine learning approaches are classified into supervised (classification and regression), unsupervised (clustering), ensemble learning and reinforcement learning. However, among selected studies, we found that no article applied unsupervised ML techniques; thus, our analysis focused on supervised ML,

ensemble learning, and reinforcement approaches. One possible reason for the prevalence of supervised ML approaches in XSS attack detection is that the problem is often framed as a classification task, where the goal is to predict whether a given input data (e.g., an XSS payload, HTTP request/response, or JS files) is malicious or benign. Supervised learning algorithms are well-suited for this type of task, as they can learn from labeled data and make predictions on new, unseen data. On the other hand, unsupervised learning techniques such as clustering are not as well-suited for this task, as they do not rely on labeled data or explicitly learn to classify data points. Clustering algorithms may still be useful in certain aspects of XSS detection, such as identifying patterns in the data or grouping similar attacks together. However, they are not as directly applicable to the classification task as supervised learning algorithms.

As shown in Table 8., the traditional ML algorithms are utilized mostly than the ensemble learning and reinforcement learning. This is likely due to the fact that the implementation and deployment of ML algorithms are generally easier than other approaches. To investigate the most used ML algorithms for XSS attack detection, we explain the principles and features of these algorithms in the following:

Artificial Neural Network (ANN): is a machine learning algorithm inspired by the structure and function of the biological brain. It is a supervised learning algorithm that can be used for both classification and regression problems. In ANN, the data is processed through a series of interconnected nodes, known as neurons, organized into layers. The neurons in each layer receive inputs and apply a set of weights to those inputs, then pass the result through an activation function to generate an output. The output of one layer becomes the input for the next layer until the final layer produces the output of the network.

ANN has been applied in various fields, such as speech and image recognition, anomaly detection, and natural language processing. In the domain of the XSS attack, ANN implementation was found in the study of (Mokbal et al., 2019), in which a novel technique for feature extraction and a Multilayer Perceptron (MLP) was proposed for the XSS-based attacks detection. The detection scheme was trained and tested on a large dataset, achieving high accuracy in detecting attacks. The study demonstrates the effectiveness of ANN in detecting complex attacks in web applications.

Decision Trees (DTs): is a popular classification method used in supervised learning and widely used in various domains. The tree-like representation of DT models' choices and outputs makes them an easily interpretable and implementable classification method. In DTs, a trained model uses packet features to determine the class of the packet. The best DT model can fit as much information as possible into as few levels as possible. Several algorithms, such as ID3, C4.5, and CART, have been proposed to generate optimal trees. These algorithms use different metrics to measure DTs' performance, such as information gain (entropy), information gain ratio, and Gini impurity. DTs are widely used in intrusion detection because of their superior generality achieved by post-construction pruning. The problem with DTs is that they aren't very durable; little modifications to the training data may lead to a very different DT. In addition, information gain favors multi-level features, may be affected by sampling error, and provides a locally optimum solution rather than the ideal answer across all possible contexts. In the study of (Kareem Thajeel et al., 2023), a set of classifiers including DT, RF, SVM, KNN, and logistic regression were used to evaluate the feature selection decisions of a multi-agent RL-based dynamic feature selection model for XSS attack detection. The DT classifier was found to obtain the best results among the other models. DT was tuned with parameters of 15 estimators, and the random generator was initialized with 42. In another study of (Panigrahi et al., 2022), a hybrid of Decision Tree and Naive Bayes models were combined to train and detect intrusions, including XSS attacks and other IDS attacks.

Random forest (RF): is a supervised machine learning algorithm that can be used for classification and regression problems. The algorithm creates a forest of decision trees, where each tree is trained on a randomly sampled subset of the data. Then it aggregates the predictions of each individual tree to generate a final prediction (Geetha & Thilagam, 2021). As the number of trees in the forest increases, the algorithm tends to perform better and produces more accurate results. RF has the same metrics (e.g.,

IG, Gini impurity, and variance) as the DTs that have to measure its performance. RF was proposed to address the problem of overfitting and improve the accuracy of DTs. DTs are prone to overfitting, which occurs when a model is trained too well on the training data and becomes too specific to that data, leading to a poor generalization of new data. In addition to its accuracy, one of the benefits of the random forest algorithm is that it can handle large datasets with many features, and it can also handle missing data (Ray, 2019). Moreover, it can be used to extract feature importance, which can be helpful in understanding the most relevant features for a given problem.

From our literature review, we found several studies that have used the RF algorithm for detecting and mitigating XSS attacks. In the study of (Lu et al., 2022), a fusion verification method was proposed by using a combination of traffic detection with XSS payload detection, where the RF classifier was used to classify the traffic and payload based on extracted and proposed features. Another work of (V. K. Malviya et al., 2021) proposed a web browser that utilizes machine learning classification to detect and mitigate XSS attacks. The browser extracts feature from web pages and JS scripts using an open-source browser (WebKit). A set of ML classifiers, including SVM, NB, RF, and ADTree, are used to classify the web pages as malicious or non-malicious. Results showed that the RF classifier using script code features achieved the best performance, with 99.5% accuracy, 99.82% precision, 99.74% recall, and 99.8% F1-score. Finally, in the study of (H.-C. Chen et al., 2021), three approaches for predicting XSS attacks in real-time were implemented. Then their results revealed that the RF algorithm performed slightly better than SVM, K-NN, and LR algorithms, achieving high accuracy, recall, and precision values.

Extreme Gradient Boosting algorithm (XGBoost): XGBoost is a highly efficient and accurate distributed gradient boosting framework designed for solving various data science problems (T. Chen & Guestrin, 2016). It is designed to handle missing values, regularization, parallelization, and distributed computing. XGBoost is a scalable, flexible, and portable algorithm that has won several machine-learning competitions. XGBoost is based on the gradient boosting framework and builds an ensemble of weak prediction models (decision trees) in a sequential manner, with each new model minimizing the loss function of the previous models. XGBoost also provides feature importance scores and feature selection capabilities. Overall, XGBoost is a powerful and widely used algorithm for classification and regression tasks. In the study by (Rozi et al., 2022), XGBoost was used as an XSS detection model. (Mokbal et al., 2020, 2021) proposed a web-based XSS attack detection framework that uses XGBoost with an extreme parameters' optimization approach using the Grid-search technique. The XGBoost algorithm has been shown to be effective in detecting XSS attacks in web applications.

Support Vector Machine (SVM): is a powerful supervised machine learning algorithm that can be used for both classification and regression tasks (Cortes & Vapnik, 1995). It is particularly effective in solving classification problems, and it has the ability to perform both linear and non-linear classification through the use of the kernel trick to map the data into a higher dimensional space where the classes can be separated using planes or other functions. Overall, SVM is a popular and highly effective classifier in machine learning due to its ability to handle complex datasets and produce accurate results.

The SVM algorithm was found to be used in the studies of (Alazab et al., 2022; Ngoc & Mimura, 2021; Phung & Mimura, 2021) as a classifier to distinguish normal and malicious attack samples. The study of (Z. Cheng et al., 2021) aimed to compare the performance of two types of models for XSS attack detection, one using the conventional feature extraction method and the other trained within the semantic structure. Four machine learning algorithms were used in the experiment, namely RT, DT, SVM, and K-neighbors The results showed that using the semantic structure significantly improved the performance of the detection model across all four algorithms, with **the SVM model having** the most dramatic improvement in F1-score, by 13%. In another study of (V. Malviya et al., 2018), a plugin-based feature extraction framework was proposed for semi-structured and unstructured data from web pages, scripts, and emails. The study used classification experiments with two datasets of script-based and web page-based features. For both datasets, a set of classifiers were used, such as RF, DT, SVM, and KNN, for classification. The SVM classifier achieved the best performance in both datasets, with an F1-score of 0.981 and 0.954, respectively. However, SVM's performance degrades with larger

datasets because of the longer training time required (Ray, 2019). Finding a useful kernel function will be challenging. In the presence of noise in the dataset, SVM performs poorly. When it comes to probability estimations, SVM is useless. It's not easy to make sense of the final SVM model.

4.3.2 Ensemble learning

Ensemble learning is a machine learning technique that aims to enhance classification performance and generalizability by training multiple base learners and combining their predictions (Ribeiro & dos Santos Coelho, 2020). This approach is a variant of machine learning and has shown promise in improving model accuracy and robustness. The fundamental idea behind ensemble learning is that machine learning models have limitations and can make errors. By combining the strengths of multiple base models, ensemble learning aims to improve classification accuracy and reduce the variance and bias errors associated with individual models (Mienye & Sun, 2022). Ensemble learning methods are broadly categorized into three categories: boosting, bagging, and stacking (Ribeiro & dos Santos Coelho, 2020).

Boosting algorithms: such as Gradient Boosting, XGBoost, and AdaBoost, are iterative algorithms that train base learners sequentially, with each new learner correcting the errors of the previous one (Sagi & Rokach, 2018). Boosting algorithms are effective at reducing bias errors and are particularly useful for classification tasks that involve imbalanced datasets. In the work of (Ghaleb et al., 2022), boosting ensemble learning is employed for XSS attack detection through using three RF and MLP for final decision-making. In another study by (Li et al., 2020), a boosting method was proposed that consists of Co-Forest, KNN, and Random Tree algorithm as the base classifier.

Bagging algorithms: such as Random Forest and Extra Trees classifier, are based on the idea of bootstrapping, which involves randomly sampling the training data with replacement to generate multiple training sets (Nti et al., 2020). Each training set is used to train a separate base learner, and their predictions are combined to make a final decision. Bagging algorithms are effective at reducing variance errors and are particularly useful for classification tasks that involve high-dimensional data. The bagging ensemble learning method is adopted to detect XSS attacks in the study of (P. Nagarjun & Ahamad, 2020) by using classifiers including RF, AdoBoost, SVM, Extra-trees, gradient boosting, and histogram-based gradient boosting. Also, the study of (Zhou & Wang, 2019), used a set of Naïve Bayes algorithms as a bagging ensemble learning method.

Stacking algorithms: such as Super Ensemble and Blending, involve training multiple base learners and using their predictions as inputs to train a higher-level model called a meta-learner (Ribeiro & dos Santos Coelho, 2020). The meta-learner combines the predictions of the base learners to make a final decision. Stacking algorithms are effective at reducing both variance and bias errors, and they are particularly useful for complex classification tasks that involve diverse types of data. (Tama et al., 2020) employed stacking ensemble learning by using several classifiers, including RF, Gradient Boost, and XGBoost.

4.3.3 Reinforcement learning

Reinforcement learning is a type of machine learning algorithm that focuses on learning through interaction with the environment. As stated in (Barto, 2018), it involves mapping input situations to output actions that maximize a measurable reward or reinforcement signal. This approach is often used in a reward-based system, which works as a semi-supervised approach to learning.

In the field of cybersecurity, reinforcement learning has been used in the detection of XSS attacks. In particular, RL has been used in various studies for generating adversarial samples to improve the robustness of existing XSS detection models. This is achieved by evolving the model or the process of sample generation through updating mutation rules to generate new adversarial examples from current XSS attack samples. The mutation rules are escaping strategies, including obfuscation, encoding techniques, replacing the alphabet style, lowercase/uppercase, removing/substituting the HTML or

JavaScript attributes/functions/ events/tags/symbols/expression, adding/removing the blank spaces, etc. In the study of (Fang et al., 2019), a DDQN RL-based Adversarial XSS attacks generation model was proposed, and the new generated samples were used against some of the Black-box and White-box detection models. The bypassed samples were then used for retraining the XSS classifier model. Similar approaches were implemented in the studies of (Q. Wang et al., 2022), which used the soft Q-learning algorithm, (L. Chen et al., 2022), which adopted the Soft Actor-Critic (SAC) RL algorithm, and (Lee et al., 2022), which used the Advanced Actor-Critic (A2C) RL. These studies demonstrated the effectiveness of RL-based approaches in generating sophisticated adversarial samples and improving the accuracy of XSS detection models. Another approach is proposed by (Tariq et al., 2021), which uses a combination of RL and Genetic algorithm (GA) to detect novel XSS attacks. This approach showed promising results in detecting novel attacks and demonstrated the potential of combining RL with other algorithms to improve the accuracy of XSS detection.

4.3.4 Deep learning

Deep learning (DL) is a subclass of machine learning that has gained popularity in recent years owing to its capacity to learn complicated representations of data. Computer vision, NLP, voice recognition, and anomaly detection are just a few domains where DL methods have been successfully utilized. (Geetha & Thilagam, 2021). Overall, deep learning models have many hierarchical layers and a highly configured structure with a number of hidden layers to enable them to identify complex patterns and relationships within the data. Each layer in a deep neural network contains numerous artificial neurons, each with its own set of weights and activation functions. The weights are used to determine the contribution of each input to the output of the neuron. During initialization, these weights are set randomly, or the same weight can be set across all the weights. However, the model will adjust these weights based on the error value to improve the model's performance. Deep learning's exceptional feature is that it can extract and abstract features automatically, eliminating the need for laborious and tiresome feature extraction and thus automatically finding complex as well as beneficial high-order features. As can be seen in Table 8., the deep learning approaches are the most used for XSS attack detection with 26 studies. Through analysis, we found that the DL algorithms used with XSS attack detection are CNN, DNN, GRU, GCN, GAN, Autoencoder, LSTM, Bi-LSTM, RNN, Bi-RNN, and hybrid models.

Convolutional Neural Networks (CNNs): is an artificial neural network (ANN) that is designed to extract translation equivariant responses called feature maps by sliding the convolutional kernel over input features. CNNs consist of three layers: convolutional, pooling, and fully connected. The convolutional layer computes different feature maps, and the pooling layer reduces the size of convolved features while maintaining important features. CNNs use different activation functions, ReLu, Maxout, Tanh and Sigmoid, to introduce the nonlinearity required to solve nonlinearly separable problems/ features. The fully connected layer determines the association between features and classifies the target classes using the softmax function (Alzubaidi et al., 2021). CNNs have shown excellent performance in computer vision and intrusion detection systems (Dong et al., 2019), reducing computational complexity and improving training and prediction speed by extracting local features. In the security domain, CNNs have been successfully applied by several studies to detect XSS attacks by proposing CNN models with different network structures and hyperparameters. Through this review, we found that the CNN model is the most famous and commonly adopted algorithm for XSS attack detection with 8 studies including (Abaimov & Bianchi, 2019; Akaishi et al., 2019; Chaudhary et al., 2021; Kuppa et al., 2022; Maurel et al., 2022; Shahid et al., 2022; Tekerek, 2021; G. Zhang et al., 2019). In addition, another 2 studies (Z. Liu et al., 2021; H. Yan et al., 2022) applied the pretraining CNN-based models called Residual Network (ResNet) (He et al., 2016). Finally, another 2 studies by (Huang et al., 2021; J. Yang et al., 2020) applied a CNN-based text classification model called TextCNN (Kim, 2014).

Deep Neural Networks (DNNs): DNNs are artificial neural networks with numerous hidden layers between the input layer and output layers of the network. These networks, which have more layers than conventional artificial neural networks (ANNs), are used to handle complex nonlinear issues. A fully

connected DNN's last-layer neurons can establish connections with all previous-layer neurons. The model receives an input vector, computes them in the hidden levels, and transmits the result to the output layer. The network's weights and biases are repeatedly fine-tuned using techniques such as stochastic gradient descent to reduce error or costs (Samek et al., 2021). Backpropagation is used with DNNs to execute guided learning tasks with nonlinear activation functions because backpropagation conducts a reverse (backwards) run after each forward transmission through the network to dynamically modify the model's parameters. DNNs are used for DOM-based XSS attack detection in the study of (Melicher et al., 2021). The authors proposed a DNN model for learning and classification with parameters set with embedding sizes to 64, 256 and 1024 for 3- hidden layers, fully connected DNN with 100, 50, and 25 units. The Adagrad is used as an optimizer with a learning rate of 0.05 and batch size of 64. However, such work is Focused only on DOM-based XSS attack detection and requires high time for training.

Recurrent Neural Networks (RNNs): RNNs are a widely used algorithm in the DL field, particularly in speech processing and NLP contexts (Hewamalage et al., 2021). RNNs were proposed to solve the problem of Deep Neural Networks' difficulty in fitting data that changes over time. Unlike traditional neural networks, RNNs utilize sequential data in the network, allowing for valuable information to be extracted from the embedded structure of the data sequence. This feature is fundamental to a range of different applications, as understanding the context of a sentence is crucial to determining the meaning of a specific word within it, as explained in the previous, section 4.2.2.4. RNNs can be considered as a unit of short-term memory, with the input layer (x), output layer (y), and state (hidden) layer (s). In the study of (Mohammadi & Namadchian, 2020) RNN-based method was proposed, in which an attention-based RNN network is used to encode the request-response data after transforming them into a numerical representation using the Word2Vec model. Furthermore, based on the requests, the value of attention is produced. To reflect the response, these values are sent through a decoder RNN network, and the output is fed into a fully connected layer for classification.

Bi-directional Recurrent Neural Networks (Bi-RNNs): Bi-RNNs are a type of RNN that can process sequences in both forward and backward directions, allowing them to capture contextual information from both past and future inputs. Bi-RNNs have been shown to be effective in a wide range of applications, including speech recognition, machine translation, and sentiment analysis. The authors of (Fang et al., 2020) introduced a Bi-RNN model to detect email XSS attacks combined with Attention mechanism to improve the training effect of the model. In the same context, in the work of (Z. Liu et al., 2023), the XSS detection method was proposed using a single layer of a Bi-RNN network with an attention mechanism. This model is used after the code is vectorized and embedded by the Word2Vec model, and the graph features of the source code are extracted by the graph convolutional neural network (GCN). On the other hand, the second method tracks the sensitive methods or attributes, the code execution sequence for extraction. However, RNNs are not equipped with a special treatment of the activation function, making the continuous production of their partial derivatives prone to gradient disappearance or even explosion when the network has a high number of layers (Samek et al., 2021).

Long Short-Term Memory (LSTM): LSTM is a type of RNN that can train and learn long-range temporal dynamics in sequences of arbitrary length (Hochreiter & Schmidhuber, 1997). LSTM was introduced to solve the problem of gradient vanishing in traditional RNNs by using a gate function that selectively allows a portion of the information to pass through (Gao et al., 2019). The LSTM model consists of an input gate, output gate, and forget gate, which act as filters to control the information to be retained or discarded before relaying the long-term and short-term information to the succeeding cell. The gates can remove irrelevant and unwanted selected information. LSTM has become one of the most used RNN variants because it is well-suited for classification and prediction based on time-series data. LSTM has been applied to detect XSS; we found 3 studies utilized LSTM algorithm, including (Fang et al., 2018; Lei et al., 2020; Stiawan et al., 2023). Another study proposed a detection method by integrating the LSTM algorithm with the CNN model within the studies of (R. Yan et al., 2018). However, the traditional LSTM architecture cannot be trained in parallel (Bai et al., 2018), leading to costly implementation resources.

Bi-directional Long Short-Term Memory (Bi-LSTM): Bi-LSTM algorithm is a variant of the LSTM algorithm that comprises two LSTM networks (Graves & Schmidhuber, 2005). These LSTMs accept input in opposite directions, i.e., forward, and backward. In contrast to LSTM, where the input flows in one direction, Bi-LSTM allows the input to flow in two directions (front and back) to preserve future and past information. This feature enables the network to create a context for each character in the input text based on its past and future. By preserving past and future information, Bi-LSTM has been shown to achieve high detection accuracy in various applications. The Bi-LSTM was employed in the study of (Hu et al., 2023), in which it was integrated with the CNN algorithm to propose a model for XSS attack detection.

Gated Recurrent Unit (GRU): GRU is a type of LSTM that is designed by (Chung et al., 2014) to handle long-term dependencies in sequential data. Like LSTM, GRU uses a gating mechanism to update and forget information at each time step selectively. However, GRU has fewer parameters than LSTM, which makes it faster to train and more computationally efficient. GRU has been utilized to detect XSS attacks and web application attacks. In the study of (Niu & Li, 2020), a combination method was proposed using a CNN-GRU model for web attack detection, where the GRU captures the serialization relationship of the context in the web event sentence. In another study of (W. Yang et al., 2019), the detection model was suggested by employing a convolutional neural network to extract features from the URL's abstract level and utilizing a GRU as a pooling layer to keep the key features while maintaining the context correlation. To completely extract features at each level, a combination of different-length convolution windows is used. It is necessary to integrate the features retrieved from the convolution windows to fully use the extracted features.

Autoencoder was initially proposed as a method to pre-train deep neural networks, which was found to improve their performance on supervised learning tasks, especially when the amount of labeled data is limited. By learning a useful representation of the input data, autoencoders can help to overcome the problem of vanishing gradients, which is a common issue with deep neural networks that have many layers. Overall, the autoencoder is a type of neural network with a symmetric structure consisting of two parts: an encoder and a decoder. The encoder maps the original input to a hidden layer using an activation function, while the decoder produces a reconstruction of the input using the encoder output (Vincent et al., 2010). The goal of the autoencoder is to minimize the difference between the target and input values, penalizing the decoder for being dissimilar from the input. This is achieved by constraining the hidden layers to have smaller dimensions than the input, forcing the autoencoder to capture the underlying structure of the training data. Autoencoder was used for XSS attack detection in the study of (Y. Pan et al., 2019). To encode and reconstruct the call graph for end-to-end deep learning, a stacked denoising autoencoder was trained with the assistance of the Robust Software Modeling Tool (RMST). RMST is a dynamic analysis tool that monitors code behavior during runtime. By utilizing a low-dimensional representation of the raw features with unlabeled request data, the autoencoder detects anomalies by computing the reconstruction error of the request data.

4.3.5 Evolutionary algorithm- Genetic Algorithm (GA)

Genetic algorithms (GAs) are a type of evolutionary algorithm used to find optimal or near-optimal solutions to complex problems by mimicking the process of natural selection (Singh et al., 2021). In the context of XSS attack detection, GAs are used to optimize the performance of machine learning algorithms by selecting the best features or improving the ML learning models against new XSS attacks.

Our review reveals that several studies proposed using of GA algorithms to optimize XSS attack detection. For instance, the GeneMiner approach introduced by (C. Gupta et al., 2022) adopted an incremental genetic algorithm to detect new XSS payloads by incorporating mutation and crossover operations in the feature set. (Suleman & Awan, 2019) implemented GA to optimize ML algorithms, including Naïve Bayes, ID3, KNN, DT, and RF, by reducing the number of features selected and showed that the accuracy rate of detection increased from 76% to 94.99%. (Z. Liu et al., 2022) used a combination of fuzzy inference and GAs to generate high-quality vectors for detecting XSS vulnerability in web pages. Using GA algorithms in XSS attack detection provides several advantages,

such as finding optimal or near-optimal solutions to complex problems, evolving new XSS payloads, and optimizing ML algorithms. However, there are some limitations to using GAs, such as their high computational cost, difficulty determining the best parameters, and the need for large data.

4.3.6 Adversarial XSS attacks

Although ML/DL algorithms have demonstrated remarkable effectiveness in detecting XSS attacks, they remain vulnerable to adversarial examples attacks due to their mostly discontinuous input-output mapping. The application of deep learning in critical security domains has raised concerns about the presence of adversarial examples (L. Chen et al., 2022). Therefore, it is critical to develop defenses against such attacks and reduce their impact on the attack detection system (Szegedy et al., 2014; Yuan et al., 2019).

We observe that very few articles consider such a new trend issue with the XSS attacks. Only six studies of existing studies (L. Chen et al., 2022; Fang et al., 2019; Lee et al., 2022; Z. Liu et al., 2022; Q. Wang et al., 2022; X. Q. Zhang et al., 2020) focus on improving the detection models against the adversarial XSS attacks. These methods proposed or used mutation rules to construct a new synthetic dataset called adversarial XSS attacks from existing payload datasets by proposing or using the current escape strategies, including URL, JavaScript, and HTML escaping rules. XSS scripts may only be altered during the creation process in accordance with those guidelines; otherwise, their original semantics will be lost. In other words, adversarial attack examples can be generated by applying some of escaping rules (mutation rules) to the original examples; these generated adversarial samples are similar to the original sample, but differ slightly in structure or composition, then these samples can lead to ML/DL models' classification errors. It is worth noting that the adversarial XSS detection studies are not included in Table 9., since such methods focus on improving the detection models more than other preprocessing steps.

Although, as mentioned earlier, the six studies generated adversarial XSS attack examples with varying escape rates (ER). However, In the future, defending against adversarial attacks should be considered an additional evaluation metric that should be taken within the validation step for any ML/DL-based detection approach.

The work of (X. Q. Zhang et al., 2020) proposed a GAN-based adversarial training method named MCTS-T for defending against XSS adversarial examples. The MCTS-T algorithm considers two factors, "modification position" and "modification action," in generating adversarial examples. To improve the effectiveness of the generated examples, the study proposes an improved Upper Confidence Bounds UCB method called UCB-T in the selection stage, applies a dropout strategy in the expansion stage, and obtains feedback from a detection model in the simulation stage. The algorithm is designed as a black model, meaning that attackers do not need to know the internal configurations of the detectors to bypass the intrusion detector. The study also used the bypassing rules to modify existing XSS attacks, including hexadecimal encoding, decimal encoding, URL encoding, inserting invalid characters, and case mixtures. The original XSS attack samples and newly generated adversarial data have been verified against the suggested CNN-based detection model.

The authors (Fang et al., 2019), introduce a novel reinforcement learning-based XSS adversarial attack model (RLXSS) that aims to improve the detection model's ability to counter XSS attacks continuously. The proposed model transforms the XSS escape attack into a set of escape strategies and selects the optimal option based on the environment's state. The authors suggest four different XSS attack escape methods, including position morphological modification, sensitive word substitution, and special character addition. To continually enhance the detection model's defense against assaults, RLXSS alternates between training the detection model and the adversarial attack model. The adversarial model mines adversarial samples with XSS attack functions that can evade both black and white-box detection. These adversarial samples are then designated as malicious and used to retrain the detection model.

The work of (Q. Wang et al., 2022) proposed a Fuzz dataset generation method based on the Fuzz algorithm. An adversarial constructing model based on maximum entropy reinforcement learning. The classification model comprises MLP, SVM and LSTM algorithms, besides the black box scanners, SafeDog, and XSSChop. The adversarial samples were generated according to the escaping strategies, including HTML and JavaScript escape strategies.

An approach based on a detection phase based on well-liked XSS attack detection models and an escape phase for XSS adversarial instances produced using the Soft Actor-Critic (SAC) reinforcement learning algorithm was suggested by (L. Chen et al., 2022). Each XSS sample should be preprocessed before being included in the detection models during the detection phase. The reward and state will be entered into the adversarial model if determined malicious. The agent selects a suitable escape action from the proposed mutation methods for the escape phase depending on the condition of the present environment, and then modifies the neural network's parameters based on the reward feedback supplied by the XSS detection model. The example is altered and then entered into the detection model according to the agent's chosen escape plan. To achieve this, the procedure is continued until the detection model determines that the example is benign.

In the study of (Lee et al., 2022), a novel method of adapting attack payloads to a target-reflected XSS vulnerability using reinforcement learning (RL) is presented. The authors propose a framework called Link, which uses RL to find reflected XSS vulnerabilities in a black-box and fully automatic manner. The framework consists of an RL agent that learns an optimal policy for generating attack payloads, a crawler that collects subdomain URLs and their input parameters for testing, and an environment that consists of a web server and an attack manager. The authors evaluate the efficacy of Link on Firing-Range, OWASP, and WAVSEP benchmarks, as well as 12 real-world applications. Link outperforms existing web scanners, including Burp Suite, Wapiti, ZAP, and Black Widow, finding vulnerabilities and reducing testing campaign time.

The study of (Z. Liu et al., 2022) proposes a method for generating fuzzing test vectors using genetic algorithms to improve the accuracy of detecting cross-site scripting (XSS) vulnerabilities in web applications. The study considers the diversity of XSS attack patterns and the need to generate valid test vectors from the attacker's perspective. The proposed method, called GAXSS, uses taint tracking to generate different types of test data and integrates a modification method for bypassing security detection mechanisms in the mutation step of the genetic algorithm. The study also introduces an advanced detection method based on genetic algorithms and XSS structure analysis.

Table 9.

Preprocessing and learning methods analysis.

Author	Data cleaning			Feature extraction	Feature representation	Dimensionality reduction	Feature selection	Balanced method	Learning Method	Classifier Algorithm	Source of dataset	Dataset type	Validation process		Performance metrics	Limitations
	Removing noisy data	Decoding	Tokenization										Cross validation	Dataset splitting size		
(Maurel et al., 2022)	×	×	√	Vectorization (code2vec and word2vec)	Vector representation	×	Attention mechanism	Semi-balanced	DL	Sigmoid	OWASP cheat sheet	Synthetic dataset	N/A	-	Acc, precision, recall, F1-measure, TNR,	High false positive and negative rates
(Kuppa et al., 2022)	√	√	√	vectorization ASCII values	Binary vector matrix	Sparse dimension reduction	×	×	DL	Softmax	Payload Github, Cheat sheets and other references	Raw dataset	10 folds	60%, 20%, 20%	Acc, precision, recall, Time consuming	ASCII not efficient to represent the semantic meaning, the results is limited to high overfitting using the majority malicious
(Abaimov & Bianchi, 2019)	×	√	×	Statistical Encoding values	numerical pattern of pairs values (code, type)	×	×	×	DL	Softmax	Payload Github,	Raw dataset	×	80%-20%	Acc, precision, recall, Time consuming	High false positive and negative rates and high dimensionality
(Rozi et al., 2022)	×	×	×	Graph2vec	vector	×	SHAP Values	×	ML	XGBoost	GAWAIN	Raw dataset	×	N/A	Acc, precision, recall, F1-measure	High false positive and negative rates
(Chaudhary et al., 2021)	√	√	×	Statistical Encoding	numerical pattern of pairs values (code,	Principal Component	×	Semi-balanced	DL CNN	Sigmoid	Payload Github and	Raw dataset	10 folds	N/A	Acc, precision, recall, F1-measure, FPR,	the no. of malicious samples greater than benign samples

				values	type)	Analysis (PCA					kaggle				Time consuming	
(Z. Cheng et al., 2021)	×	×	segmentation	Contextual (Semantic)	vector representation	×	×	×	ML	RF, DT, SVM, K-NN	CSIC-2010	Synthetic dataset	×	-	Acc, precision, recall, F1-measure, FPR	very high false negative and false positive
(Niu & Li, 2020)	×	×	segmentation	Statistical Encoding values and vectorization (word2vec)	Numerical features and vector representation	×	×	×	DL (CNN and GRU)	Softmax	CSIC-2010	Synthetic dataset	×	N/A	Acc, precision, recall, F1-measure	High false positive and negative rates
(Tama et al., 2020)	×	×	×	×	Numerical features	×	×	×	ML (stacking ensemble learning)	RF, Gradient Boost, XGBoost	CSIC-2010v2, CICIDS-2017, NSL-KDD and UNSW-NB15	Secondary datasets	10 folds	80% and 20%	Acc, precision, recall, F1-measure, FPR, ROC, and AUC	Used secondary datasets, so no feature selection or reduction were used
(Akaishi et al., 2019)	×	×	segmentation	Statistical Frequency of appearance and vectorization (word2vec)	Vector	×	×	×	DL and ML (CNN, CNN+NB)	NB, and Softmax	OWASP cheat sheet	Raw dataset	5 folds	80%-20%	Acc, precision, recall, F1-measure	-

***** N/A indicates **not applicable**.

Table 9.

Preprocessing and learning methods analysis (Continue…).

Author	Data cleaning	Feature	Feature	Dimensionality	Feature	Balanced	Learning	Classifier	Source of dataset	Dataset	Validation process	Performance	Limitations

	Removing noisy data	Decoding	Tokenization	extraction	representation	reduction	selection	method	Method	Algorithm		type	Cross validation	Dataset splitting size	metrics	
(Alazab et al., 2022)	×	using BeautifulSoup parser	×	Domain-expert (handcrafted)	Numerical features	×	correlation coefficient, Information Gain (IG),	Classifier-independent	ML	SVM	Alexa Top 500 and VX Heaven (vxheaven.org).	Raw dataset	10 folds	60%-40%	Acc, precision, recall, F1-measure, FPR and Time consuming	Very view malicious samples were used in training
(Z. Cheng et al., 2020)	×	×	segmentation	Contextual (Semantics)	Vector representation	×	×	×	ML	Pattern-tree algorithm	CSIC-2010	Synthetic dataset	×	-	Acc, precision, recall	High false positive, and false negative
(Fang et al., 2018)	generalization	√	√	Vectorization (Word2vec), and Context analysis (LSTM)	Vector representation	×	×	×	DL (LSTM)	Softmax	XSSed and DMOZ	Raw dataset	10 folds	-	precision, recall, F1-measure, FPR, ROC, and AUC	the no. of malicious samples greater than benign samples
(Fang et al., 2020)	Generalization, lowering upper case, and removing disturbing characters	√	√	Vectorization (Word2vec), and Context analysis (Bi-RNN)	Vector representation	×	Attention mechanism	control the positive sample with less than 1000 characters.	DL (Bi-RNN)	Softmax	China's WooYun web mail Database and Enron Email Dataset	Raw dataset	10 folds	60%-40%	Acc, precision, recall, F1-measure, and AUC	High false negative and not balanced
(H. Yan et al., 2022)	Generalization	√	segmentation	Vectorization (Word2vec), and 2 convolutional layers with 1 layer pooling layer	Vector representation	×	×	Used balanced dataset	DL pretrained CNN model (ResNet)	Softmax	XSSed and DMOZ	Raw dataset	×	70%-30%	Acc, precision, recall, F1-measure	Their training and validation based on balanced dataset
(C. Gupta et al., 2022)	removing disturbing characters	√	×	Statistical Feature [illegible]	Binary vector representation (0,1)	×	×	×	Evolutionary algorithm	Incremental Genetic Algorithm (IGA)	Training dataset from GitHub PL[8] and duoergun[9] and Testing dataset from Zhou and Wang	Raw dataset	×	N/A	Acc, precision, recall, F1-measure, and TNR	Based on traditional statistical feature EX and Representation

[8] https://github.com/payloadbox/xss-payload-list/

[9] https://github.com/duoergun0729/1book/tree/master/data

(Hu et al., 2023)	Generalization	√	segmentation	Vectorization (Word2Vec, Skip-gram), Context analysis (Bi-LSTM), and 3 parallel convolutional layers	Vector representation	×	Self-attention mechanism	×	DL (CNN and Bi-LSTM)	Softmax	XSSed, benign from https://curlie.org/en	Raw dataset	10 folds	90%-10%	Acc, precision, recall, F1-measure	Not consider the efficiency of the model, requires high dataset for training

***** N/A indicates **not applicable**.

Table 9.

Preprocessing and learning methods analysis (Continue…).

Author	Data cleaning			Feature extraction	Feature representation	Dimensionality reduction	Feature selection	Balanced method	Learning Method	Classifier Algorithm	Source of dataset	Dataset type	Validation process		Performance metrics	Limitations
	Removing noisy data	Decoding	Tokenization										Cross validation	Dataset splitting size		
(Huang et al., 2021)	×	×	each line of AST used as syntactic unit (SU) with 187 different SU	Vectorization (Word2Vec), Context analysis (2 Bi-LSTM layers)	Vector representation	×	×	Used balanced dataset	DL (TextCNN)	Softmax	Alexa Top 100, and Github (HynekPetrak, 2017), malicious platform VX Heavens (VX, 2020) and Curtsinger et al. (2011).	Raw dataset	5 folds	80%-20%	Acc, precision, recall, F1-measure, AUC, and Time consuming	Their training and validation based on balanced dataset, required long time for training and detection
(Ghaleb et al., 2022)	Removing the unwanted words and characters	×	√ Tokenization	Word and character N-gram,, each one of them is applied with different dataset, and statistical-based text representation	Vector representation	×	Information Gain (uses entropy to measure impurity)	×	ML (ensemble learning)	Three RF, and MLP for final decision making	Kaggle[10], Phishtank[11], and ISCX-URL dataset[12]	Raw dataset	×	70%-30%	Acc, precision, recall, F1-measure, and FPR	High dimensional features

[10] https://www.kaggle.com/sid321axn/malicious-urls-dataset

[11] https://phishtank.org/

[12] https://www.unb.ca/cic/datasets/url-2016.html

(H. Pan et al., 2022)	lowercasing, and generalization	√	Segmentation	Vectorization Word2Vec (CBOW), and Convolutional-based feature extraction (GCN)	Graph representation	×	Point-wise mutual information (PMI)	×	DL	GCN	GitHub Repository[13], and Kaggle platform[14]	Raw dataset	×	60%:20%:20%	Acc, precision, recall, F1-measure, FPR, and Time consuming	-

***** N/A indicates **not applicable**.

Table 9.

Preprocessing and learning methods analysis (Continue…).

Author	Data cleaning			Feature extraction	Feature representation	Dimensionality reduction	Feature selection	Balanced method	Learning Method	Classifier Algorithm	Source of dataset	Dataset type	Validation process		Performance metrics	Limitations
	Removing noisy data	Decoding	Tokenization										Cross validation	Dataset splitting size		
(Lei et al., 2020)	Generalization	√	Segmentation	Vectorization (Word2Vec, Skip-gram), Context analysis LSTM layers	Vector representation	×	×	×	DL (LSTM)	Softmax	Xssed and DMOZ	Raw dataset	×	70%-30%	precision, recall, F1-measure	High fasle positive and negative rates
(Li et al., 2020)	~~Lowercasing~~	√	×	~~Domain~~-expert (handcrafted)	Numerical features	-×	WEKA using selection algorithm CfsSubseEval and greedy search	Threshold valude added to each classifier by	ML (ensemble learning)	Co-Forest, KNN, and Random Tree algorithm as the base	Benign samples from the traffic flow of Bejing university.	Synthetic dataset	×	×	Acc	High fasle positive and negative rates

[13] https://github.com/duoergun0729/1book/tree/master/data

[14] https://www.kaggle.com/datasets/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning

							algorithm	measuring ratio of positive to negative samples		classifier	And malsious from XSSed					
(Z. Liu et al., 2021)	generalization	√	segmentation	Word2Vec, and GCN	Graph representation	×	×	×	DL (GCN and Residual network block)	SoftMax	XSSed and normal HTTP GET request from the laboratory servers	Raw dataset	10 folds	×	Acc, recall, F1-measure, and AUC	-
(Z. Liu et al., 2023)	generalization	×	segmentation	Vectorization Word2Vec (CBOW), Contextual (Bi-LSTM) Convolutional-based feature extraction (GCN), CFG	Multi-feature Graph and vector representations	×	Self-Attention (weighted aggregation)	Used the majority of class label is positive samples	DL (GCN, Bi-RNN, and MLP)	SoftMax	XSSed, benign from high-traffic websites published by Similarweb (2022)	Raw dataset	10 folds	70%-10%-20%	Acc, precision, recall, and F1-measure	Performance was affected when test with mutated samples, time consuming with around 200 epoch
(V. Malviya et al., 2018)	-×	×	×	Statistical features	Feature vector	×	×	×	ML	SVM, Naïve byes, RF, ADTree	XSSed and Alexa top-500 sites.	Raw dataset	10 folds	N/A	Acc	Complicated feature extraction tool that's required a write specific code for each feature.
(V. K. Malviya et al., 2021)	×	√	×	Statistical features	Numerical features	×	×	×	ML	SVM, Naïve byes, RF, ADTree	XSSed and Alexa top-500 sites.	Raw dataset	10 folds	N/A	Acc, precision, recall, F1-measure, and Time consuming	Complicated feature extraction tool that's required a write specific code for each feature

***** N/A indicates **not applicable**.

Table 9.

Preprocessing and learning methods analysis (Continue…).

Author	Data cleaning			Feature extraction	Feature representation	Dimensionality reduction	Feature selection	Balanced method	Learning Method	Classifier Algorithm	Source of dataset	Dataset type	Validation process		Performance metrics	Limitations
	Removing noisy data	Decoding	Tokenization										Cross validation	Dataset splitting size		
(Melicher et al., 2021)	Removing duplicated script codes	×	segmentation based fixed-semantic-distance strategy	Vectorization (CBOW), expert domain	Vector representation	×	Oversampling for frequently appearing functions	Average weight to positive function during training	DL (DNN)	-	The authors collect the data by performing a large-scale web crawl of Alexa top 10,000	Raw dataset	5 folds	80%-10%-10%	Acc, precision, recall, and Time consuming	Focus only on DOM-based XSS attack detection
(Mohammadi & Namadchian, 2020)	×	×	×	Vectorization word2vec	Vector representation	×	Attention	×	DL (RNN)	Softmax	two datasets including HTTP DATASET CSIC 2010 [12] and CICIDS2017 [13].	Synthetic dataset	N/A	N/A	Acc, precision, recall, F1-measure, and TNR	not effective with XSS attack detection
(Mokbal et al., 2019)	Based on using server parsers namely, BeautifulSoup library, html5lib parse, Esprima.		×	Statistical features	Numerical representation	×	×	×	ML (ANN-based MLP)	sigmoid	data is collected by performing web crawl of Alexa top 50,000, XSSed and Open Bug Bounty.	Raw dataset	10 folds	80%-20%	Acc, precision, recall, F1-measure, FPR, ROC and AUC	insufficient feature size was used
(Mokbal et al., 2020)	×	×	×	×	Numerical representation	×	×	Oversampling the minority class using conditional GAN Wasserstein GAN with a gradient penalty (C-WGAN-GP)	ML	XGBoost	Two datasets were used CICIDS2017 and (Mokbal et al., 2019)	Secondary	10 folds	70%-30%	Precision, recall, F1-measure, TNR, DR, and AUC	Focused on addressing the unbalanced challenge of the dataset, thus, secondary datasets was used

(Mokbal et al., 2021)	Based on using server parsers namely, BeautifulSoup library, html5lib parse, Esprima.		×	Statistical features	Numerical representation	Based on feature selection	Information Gain (IG) fusing with Sequential backward selection (SBS	×	ML	XGBoost	Data was collected by performing a large-scale web crawl of Alexa top 50,000, XSSed and Open Bug Bounty.	Raw dataset	10 folds	60%-20%-20%	Acc, precision, recall, F1-measure, FPR, AUC, and Time consuming	-

***** N/A indicates **not applicable**.

Table 9.

Preprocessing and learning methods analysis (Continue…).

Author	Data cleaning			Feature extraction	Feature representation	Dimensionality reduction	Feature selection	Balanced method	Learning Method	Classifier Algorithm	Source of dataset	Dataset type	Validation process		Performance metrics	Limitations
	Removing noisy data	Decoding	Tokenization										Cross validation	Dataset splitting size		
(P. Nagarjun & Ahamad, 2020)	×	×	Unicode integer format, and standardization	Statistical features	Numerical representation	×	×	Used balanced dataset	ML (ensemble learning/bagging)	RF, AdoBoost, SVM, Extra-trees, gradient boosting, histogram-based gradient boosting	Static XSS detection tools used to collected samples for different resources	Synthetic dataset	5 folds	80%-20%	Acc, precision, recall, and F1-measure,	not used the decoding, used fixed length of payload vector to 128, which this leads to missing more significant information
(Y. Pan et al., 2019)	×	×	Bag of features with RSMT tool	Graph representation	graph	PCA	×	×	DL (autoencoder)	N/A	Use web traffic simulator to collect different samples of	Synthetic dataset	N/A	N/A	Acc, precision, recall, F1-measure, and Time	Not focused on XSS attacks only but also SQL, and remote

											different attacks, but the XSS attack collected from XSSed.				consuming	attacks. Low performance with XSS attack detection
(Panigrahi et al., 2022)	×	×	×	×	Numerical representation	Based on feature selection	Multi-Objective Evolutionary Feature Selection (MOEFS)	Down sampling (under sampling)	ML	combination of DT, and NB	CICIDS2017	Secondary	10 folds	66%-34%	Acc, precision, recall, FPR, DR and AUC	Based on secondary dataset, the XSS samples is few in comparison with other attacks' samples
(Lu et al., 2022)	Lowercasing the code	√	×	Statistical features	Numerical representation	×	×	×	ML	RF	CVE from NVD[15] and GitHub[16]	Raw dataset	20 folds	70%-30%	Acc, precision, recall, and F1-measure	focused on detecting the Reflected XSS and Stored XSS attacks and ignores the DOM-based XSS.
(Tekerek, 2021)	removing noisy data, missing values	×	×	Vectorization using (BoW) and statistical used the counting of the frequency of BoW	numerical feature vector	×	×	Used the majority of class label is positive samples	DL (CNN)	Sigmoid	CSIC2010v2	Synthetic dataset	×	70%-30%	Acc, precision, recall, F1-measure, FPR, TNR, and AUC	Use the majority class is the malicious sample

Table 9.

Preprocessing and learning methods analysis (Continue…).

Author	Data cleaning			Feature extraction	Feature representation	Dimensionality reduction	Feature selection	Balanced method	Learning Method	Classifier Algorithm	Source of dataset	Dataset type	Validation process		Performance metrics	Limitations
	Removing noisy data	Decoding	Tokenization										Cross validation	Dataset splitting size		

[15] https://nvd.nist.gov/

[16] https://github.com/duoergun0729/1book/tree/master/data

(Stiawan et al., 2023)	Removing noisy data	×	×	Contextual analysis (LSTM)	Numerical representation	PCA	×	×	DL (LSTM)	Softmax	Comnet Laboratory portal dataset[17]	Synthetic dataset	×	80%-20%	Acc, precision, recall, F1-measure, AUC, and time consuming	High false positive rate and time consuming
(R. Yan et al., 2018)	Based on using Esprima.paraser for cleaning and decoding		×	Vectorization Tri-gram, Convolutional-based feature extraction, and Contextual (LSTM)	Feature vector	×	information gain, Chi-square test and document frequency	×	DL (CNN+LSTM)	Softmax	N/A	Raw dataset	5 folds	N/A	Acc, precision, recall, , AUC, and time consuming	it costs time in the training and test phase
(J. Yang et al., 2020)	×	×	Segmentation based regular expression	Vectorization (word embedding), and Contextual (Bi-LSTM)	Feature vector	×	Attention	×	DL (TextCNN)	Softmax	Web requests from CSIC and CTF	Raw dataset	×	64%-36%	Acc, precision, recall, F1-measure	High dimensionality with size of 400
(W. Yang et al., 2019)	Remove the duplicated samples	×	Segmentation at character level	Convolutional-based feature extraction, and Contextual (GRU)	Vector representation	×	×	Balanced samples for each attack type	DL (CNN+GRU)	Softmax	well-known Chinese Internet security company,	Raw dataset	×	60%-20%-20%	Acc, precision, recall, F1-measure	High dimensionality with size of vector of 128*200
(G. Zhang et al., 2019)	normalization	√	segmentation	Vectorization (word2vec (skip-gram)), Convolutional-based feature extraction,	Vector representation	×	×	×	DL (CNN)	Softmax	XSS attack packet traffics are generated using proposed rules	Synthetic dataset	×	70%-30%	FPR and DR	High dimensionality and false positive and negative rates
(Zhou & Wang, 2019)	Lowercasing	√	×	Statistical features and domain knowledge	Numerical features	×	×	×	ML (ensemble learning-Bagging)	Set of Naïve Bayes	Training dataset from GitHub[18] and Testing dataset from various GitHub[19] and security	Raw dataset	×	10 times repeated with different percentage of malicious XSS	Acc	Few features and based only on accuracy metric in their performance evaluation

[17] https://github.com/comnetslabunsri/datasets

[18] https://github.com/duoergun0729/1book/tree/master/data

[19] GitHub - IramTariq/XSS-attack-detection

											references			records		

* N/A indicates **not applicable**.

Table 9.

Preprocessing and learning methods analysis (Continue…).

Author	Data cleaning			Feature extraction	Feature representation	Dimensionality reduction	Feature selection	Balanced method	Learning Method	Classifier Algorithm	Source of dataset	Dataset type	Validation process		Performance metrics	Limitations
	Removing noisy data	Decoding	Tokenization										Cross validation	Dataset splitting size		
(Tariq et al., 2021)	×	×	×	Statistical features and domain knowledge	Binary representation	×	RL to select/modify the relevant pattern based on classification error rate	×	ML (GA with RL)	Distance based	1st dataset from GitHub[20] and security references And 2nd dataset from Fang et al. (2018)	Raw dataset	×	10 times repeated with different percentage of malicious XSS records	Acc, precision, recall, F1-measure and time consuming	Few features, no clear preprocessing task implementation
(Shahid et al., 2022)	×	×	×	Statistical features and Convolutional-based feature extraction,	Vector representation	×	×	Over-sampling (SMOTE)	DL (CNN)	Softmax	First dataset is collected by using web scanners, The second is CSIC dataset	Raw dataset and synthetic dataset	×	N/A	Acc, precision, recall, F1-measure	High dimension with embedding layer size of 19407, 50438, 10
(Kareem Thajeel et al., 2023)	×	×	×	Statistical features and domain knowledge	Numerical features	Based on feature selection	Dynamic feature selection based multi-agent deep Q-network	×	ML	Set of classifiers including RF, DT, SVM, KNN, and logistic regression	dataset are orignially collected from XSSed, XSSply, Open Bug Bounty. Top Alexa 50,000	Secondary datasets	×	80%-20%	Acc, precision, recall, F1-measure, and memory consuming	This approach is validated with only secondary datasets

[20] GitHub - IramTariq/XSS-attack-detection

											websites.					
(Ngoc & Mimura, 2021)	Remove the duplicated samples	×	×	Vectorization (Doc2Vec)	Vector representation	×	×	Oversampling	ML	SVM	malicious URLs from PhishTank and normal from Top Alexa's websites	Raw dataset	10 folds	-	Acc, precision, recall, F1-measure	The model has a low recall rate that indicates the model is missing many relevant instances.
(Phung & Mimura, 2021)	Remove the duplicated samples	×	×	Vectorization (Doc2Vec)	Vector representation	×	×	under sampling	ML	SVM	malicious URLs from PhishTank and Github[21], while the normal from Top Alexa's websites	Raw dataset	10 folds	50%:50%	Acc, precision, recall, F1-measure, and time consuming	The model has a low recall rate that indicates the model is missing many relevant instances.

[21] https://github.com/HynekPetrak/javascript-malware-collection

4.4 Datasets

Through our review, we examined the XSS attack datasets that were used in the selected studies. Then, we categorized the most used datasets according to their dataset types into Raw, Secondary, and Synthetic datasets. These datasets (raw, secondary, and synthetic datasets) involve different data types for XSS detection, including XSS payload, JS files, and HTTP requests/responses. Usually, the JS files are extracted by scraping the contents of web pages. On the other hand, the payload can be extracted from the URL and HTTP requests/responses from the browser and server, respectively. Thus, the methods trained with XSS payload can be implemented on the browser or server side. The significant difference between the XSS payload and the JS file is that the payload is just single-line script code embedded within web pages or URL components. In contrast, the JS file is multiple script code lines written only using JS programming language.

- **Raw dataset:** is an original and real sample that needs to be cleaned and preprocessed to extract the features before being fed to the learning model.
- **Secondary dataset:** This dataset can be defined as a preprocessed dataset consisting of several extracted features and is ready to be used with AI-based XSS detection methods after certain cleaning and preprocessing steps. However, such datasets are lacking in the domain of XSS attacks.
- **Synthetic dataset:** This is a new raw dataset that the authors generated for research purposes. Such a dataset is generated, for example, by following the OWASP cheat sheet rules or setting an environment or using certain tools to simulate the attack occurring.

Our comprehensive review of ML/DL-based detection of XSS attacks has identified the most used datasets for model validation, as depicted in Table 10., **RQ#4 (a).** The XSSed archive and Payload list-GitHub are the most used raw XSS payload datasets in 15 studies. Another four studies used different scanner tools or simulators to generate synthetic datasets from the XSSed archive. For HTTP request/response data types, the CSIC-2010 and CSIC-2017 are the most frequently used datasets in 10 studies. Additionally, only three studies utilized datasets of raw JavaScript samples for the detection of XSS and other web application attacks from these resources (HynekPetrak, 2017), (VX Vault, 2017) and (Chaiban et al., 2022). While our analysis provides insight into the most used datasets in this field, it is important to note that using limited datasets may introduce certain limitations to the effectiveness of the proposed models in real-world scenarios. Therefore, it is necessary to conduct further research with larger and more diverse datasets to ensure the accuracy and generalizability of ML/DL-based XSS detection models.

Table 10.

Most used datasets with ML/DL-based XSS attack detection.

Dataset/repository	Year	Publicly available/source	Data type	Dataset type	Studies
XSS vector cheat sheet	2019	(Cozamanis, 2019)	Payload	Raw	(Kuppa et al., 2022)
XSS evasion cheat sheet-OWASP	2018	(R. Hansen & J. Manico, 2018)	Payload	Raw	(Akaishi et al., 2019), (Maurel et al., 2022), (Lee et al., 2022)

Payload Box-GitHub	2021	(İsmail Taşdelen, 2021)	Payload	Raw	(Abaimov & Bianchi, 2019)
Payload XSS dataset-Kaggle	2020	(SHAH, 2020)	Payload	Raw	(Chaudhary et al., 2021)
Malicious platform VX Heavens	2017	(VX Vault, 2017)	JS	Raw	(Alazab et al., 2022)
Malicious JS files	2017	(HynekPetrak, 2017)	JS	Raw	(Huang et al., 2021)
XSSed Archive repository	2012-2015	(XSSed Archive, 2015)	Payload	Raw	(Fang et al., 2018), (H. Yan et al., 2022), (Hu et al., 2023), (Z. Liu et al., 2021), (Z. Liu et al., 2023), (V. Malviya et al., 2018), (Mokbal et al., 2019), (Mokbal et al., 2021), (Fang et al., 2019), (L. Chen et al., 2022)
China's WooYun web mail XSS	2020	(Wooyun, 2020)	Payload	Raw	(Fang et al., 2020)
Payload list-GitHub	2017	(Duoergun, 2017)	Payload	Raw	(C. Gupta et al., 2022), (H. Pan et al., 2022), (Lu et al., 2022), (Zhou & Wang, 2019), (Tariq et al., 2021)
ISCX-URL	2016	(ISCX-URL, 2016)	Payload	Raw	(Ghaleb et al., 2022)
Phish Tank URL		(PhishTank, n.d.)	Payload	Raw	(Ghaleb et al., 2022), (Ngoc & Mimura, 2021), (Phung & Mimura, 2021)
JS-based web pages dataset	2022	(Chaiban et al., 2022)	JS	Raw	(Rozi et al., 2022)
CSIC-2010	2010	(Carmen Torrano et al., 2010)	HTTP request/response	Synthetic	(J. Yang et al., 2020), (Z. Cheng et al., 2021), (Niu & Li, 2020), (Z. Cheng et al., 2020), (Tekerek, 2021)
CSIC-2017	2017	(I. Sharafaldin et al., 2017)	HTTP request/response	Synthetic	(Tama et al., 2020), (Mohammadi & Namadchian, 2020), (Mokbal et

					al., 2020), (X. Q. Zhang et al., 2020)
Chinese internet security dataset	2018	(Cui et al., 2018)	payload	Raw	(W. Yang et al., 2019)
Using different scanners or simulator application		(XSSed Archive, 2015)	payload	Synthetic	(Li et al., 2020), (P. Nagarjun & Ahamad, 2020), (Y. Pan et al., 2019), (Z. Liu et al., 2022)
Comnet Laboratory portal dataset	2019	(Stiawan et al., 2019)	HTTP request/response	Synthetic	(Stiawan et al., 2023)
DeepXSS dataset	2019	(Yong Fang et al., 2018)	Payload	Raw	(Q. Wang et al., 2022)

4.5 Evaluation Metrics

From the perspective of uncovering the common utilized performance metrics for evaluating the XSS attack detection approaches, we found that most metrics used in almost all selected studies are based on confusion matrix parameters. The confusion matrix parameters are true positive (TP), true negative (TN), false positive (FP), and false negative (FN). True positive (TP) represents the number of samples recognized correctly as XSS attacks. True negative (TN) signals the number of samples classified as benign. False positive (FP) denotes the number of samples misclassified as XSS attacks. False negative (FN) refers to the number of XSS attack samples undetected by the system. Thus, the commonly used metrics are accuracy, recall, precision, F1-measure, and False Positive Rate (FPR) **RQ#4 (b)**.

Accuracy (Acc): is a proportion of the samples classified correctly to the total number of classified samples. This accuracy rate is computed by Eq. (1).

$$Acc = \frac{TP + TN}{(TP + TN + FP + FN)} \quad (1)$$

Recall (Sensitivity, True positive rate (TPR), or Detection rate): is a metric that denotes the ratio of the number of positive samples that were correctly predicted (TP) to the overall classifications number in the same actual class. In other words, the percentage of XSS attacks classified correctly to the total number of inputs XSS attack samples. Recall metric is used to demonstrate the FN of the model. The Recall can be calculated by Eq. (2).

$$Recall = \frac{TP}{(TP + FN)} \quad (2)$$

Precision: represents the proportion of numbers of the samples that are predicted correctly to the overall number of positive samples. Such a metric is used to show the percentage of the FP rates and can be computed using the following Eq. (3).

$$Precision = \frac{TP}{(TP + FP)} \quad (3)$$

F1-measure: is a measure that combines the recall and precision metrics in a single measure. F1-measure is also called a harmonic mean of the earlier two metrics. The F1-measure is mostly used as an evaluation metric for imbalanced datasets and can be calculated by Eq. (4).

$$F1 - measure = 2 \times \frac{(Recall \times Precision)}{(Recall + Precision)} = \frac{2TP}{2(TP + FP + FN)} \quad (4)$$

FPR (False Positive Rate): It is the ratio of all benign samples which are incorrectly detected as malicious as computed in Eq. (5). FPR was the most often used performance measure in XSS attack detection; lower FPR indicates better detection performance.

$$FPR = \frac{FP}{TN + FP} \quad (5)$$

Other: Other metrics were rarely used for evaluating the performance of the proposed detection models. Such metrics include Area Under the ROC Curve (AUC), Escape rate (ER) that used in adversarial XSS detection evaluation, time-consuming that involves the required time for training, testing or detection, memory consumption that represents the memory space required for storing the model learning, etc. AUC and ER are computed in Eq. (6-7).

$$AUC = \frac{1}{2}(\frac{TP}{TP + FN)} + \frac{TN}{TN + FP}) \quad (6)$$

Escape Rate (ER): denotes the percentage of malicious samples identified as benign by the detection model. More vectors representing escape suggest more defect in the defensive model in response to an adversarial attack; hence a high ER means greater defect.

$$ER = \frac{No.\ of\ escaping\ success}{Total\ number\ of\ malicious\ samples} = 1 - DR \quad (7)$$

Our systematic analysis of ML/DL-based XSS attack detection revealed that the evaluation of XSS detection performance is often based on common performance metrics such as Accuracy (Acc), Recall, Precision, and F1-measure, as depicted in Fig. 7. These metrics have been used frequently in 44, 43, 45, and 38 studies, respectively, indicating their widespread use and reliability in evaluating XSS detection models. The AUC metric was used in 12 studies, while FPR was used in 13 studies. In terms of computational efficiency, time consumption was reported in 16 studies, whereas memory consumption was considered in only one study. These results suggest that the evaluation of XSS detection models is mainly based on standard performance metrics, which have demonstrated their effectiveness in detecting XSS attacks.

Overall, our review highlights that most studies focus solely on improving the detection ability of the models by increasing the epochs or iterations of the training steps to enhance the model's performance. However, this may lead to overfitting and training time-consuming, which results in high computational complexity and resource consumption. Thus, a trade-off problem exists highly between the detection accuracy and efficiency. It is important to note that many studies mainly focus on improving the detection ability, with very few of them considering the efficiency side. Furthermore, the high dimensionality of the input data, including the HTTP request/response and JavaScript code, adds an additional challenge to the problem of XSS attack detection.

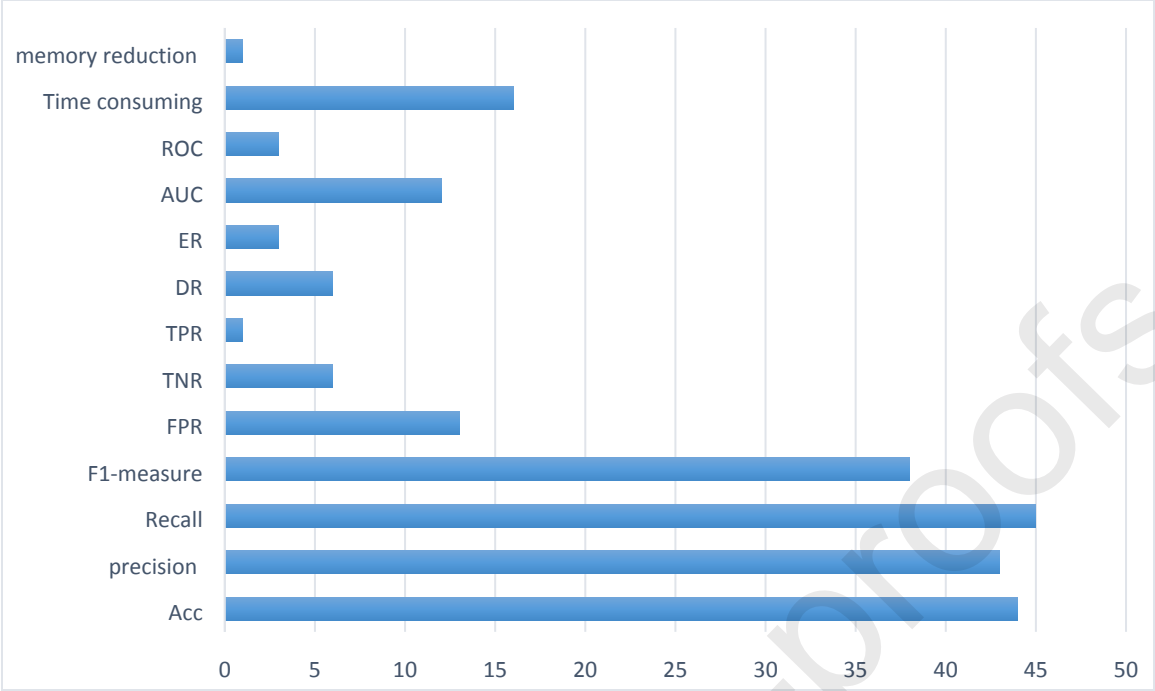

Fig. 7. Performance metrices employed in the studies.

4.6 Validation process

In ML/DL techniques, it is important to validate the model's performance to ensure it can accurately predict the target variable. Two of the most common methods for evaluating the performance of a model through the validation process are splitting dataset and cross-validation technique.

Splitting the dataset involves splitting the data into two sets for training and testing, sometimes into three sets training, validation, and testing. The training set is used to train the model, and the testing set is used to evaluate the model's performance. This process helps to ensure that the model is not overfitting the training data and can generalize well to new data. The percentage of data splitting used for training and testing affects the model's performance. Typically, a larger percentage of the data is used for training, while a smaller percentage is used for validation and testing. The exact percentages used may vary depending on the dataset's size and the model's complexity. Another commonly used technique in machine learning for model validation is cross-validation. Cross-validation involves partitioning the data into several subsets K-fold randomly and using each subset (fold) as a testing set while the remaining subsets are used for training (Refaeilzadeh et al., 2009). This process is repeated for each subset (fold), and the results are then averaged to obtain a more accurate estimate of the model's performance. Overall, validation processes are critical in ensuring that ML/DL models are accurate and reliable. It is important to carefully consider the methods used for validation and the percentage of data used for training and testing to obtain the best possible results.

Through the current systematic literature review of validation methods in ML/DL-based XSS attack detection, we analyzed the data splitting percentage and cross-validation techniques. Our findings revealed that cross-validation techniques were implemented in 24 of the 52 selected studies in our analysis. Among these studies, the 10-fold cross-validation method was the most frequently used, with 16 studies employing

it, while 5-fold cross-validation was used in 6 studies, and only one study divided the data into 20 folds. Regarding data-splitting percentages, only 26 studies declared their splitting percentages for training and testing data. The most used splitting percentages for training and testing were 80%-20% and 70%-30%, used in 8 and 7 studies, respectively. In contrast, 4 studies used a splitting percentage of 60%-20%-20% for training, validation, and testing, while the others used different splitting percentages.

Our results indicate that most studies focus on the commonly used cross-validation and data splitting techniques, which may limit the generalizability of the findings. Moreover, the optimal data splitting percentage and cross-validation technique may vary depending on the dataset's characteristics and the ML/DL model. Therefore, future studies should consider exploring alternative validation methods to improve the generalizability of their findings.

5 Limitation and Future work directions

In this study, we highlight several limitations of ML/DL-based XSS attack detection approaches. In addition, we suggest the potential solutions for each challenge for future works as shown in the following:

- The main limitation in the domain of XSS attack detection is the scarcity of updated labeled data for novel types of XSS attacks. This can turn out the current detection system in out date and may result in attack misclassification. Future work in this context should focus on creating and curating large-scale, high-quality labeled datasets for a wide range of XSS attack types. Active learning techniques and transfer learning methods can also participate in mitigating this challenge and improving model's performance.

- Dimensionality reduction and feature selection techniques in ML/DL-based XSS attack detection are limited in the current studies. Such limitations can result in high computational complexity and cause an overfitting issue that impacts the model's overall performance. This limitation can be addressed in the future by introducing more advanced dimensionality reduction and feature selection techniques to improve the effectiveness of ML/DL-based XSS attack detection models.

- Many studies are found to have high false positive and false negative rates that can lead to significant consequences, such as blocking legitimate user requests or allowing attackers to exploit vulnerabilities. To address these challenges, future work should focus on developing more robust and accurate ML and DL models, exploring new feature selection and extraction techniques, and integrating online learning methods to improve the overall performance of XSS attack detection systems.
- Current ML/DL-based XSS attack detection solutions have not considered the main challenge of concept drift and feature drift. Concept drift refers to the change in the distribution of XSS attacks data over time, while feature drift refers to the change of XSS features relevancy. However, the current proposed studies of ML/DL-based XSS attack detection approaches are based on batch learning techniques, which limits their ability to detect evolved or unknown attacks as they rely on pre-existing data and do not consider changes in the distribution or relevance of features over time. Furthermore, batch learning assumes that the data is static while the data and the distribution of attacks are dynamic and change over time. Hence, batch cannot provide the adaptability that is essential for real-world malicious XSS attack detection methods. Future studies need to propose concept drift and feature drift-aware approaches using dynamic learning techniques and Meta-

learning methods that can adapt to changes in the distribution and relevance of features over time, such as online-learning-based, active-learning-based, optimization-based and reinforcement learning-based techniques.

- Only six of the current studies focused on addressing adversarial XSS attacks in the last four years (from 2019 to the present). Adversarial attacks have a potential significant threat to ML/DL-based approach that requires further attention and more robust solutions to be developed to handle such attacks. We recommend developing effective detection methods against adversarial XSS attacks in future work. Additionally, there is a need for updated datasets that can represent real-world adversarial XSS attacks to aid in developing and evaluating these methods.
- There is a lack of studies on applying ML/DL techniques for XSS attack detection in the context of Industry 4.0 and 5.0 domains. Attacking the SCADA web-based application and IoT technologies can allow the attacker to remotely monitor and control industrial components. As these technologies become increasingly prevalent, it is crucial to investigate the effectiveness of ML/DL-based XSS attack detection techniques in such domains. Future research could focus on developing and evaluating new ML/DL-based approaches for XSS attack detection in these emerging areas.

6 Conclusion

In this work, a comprehensive systematic literature review of ML/DL techniques for XSS attack detection has been conducted. The study reviewed 52 highly quality studies and analyzed them from different aspects. Our analysis included exploring the various preprocessing techniques employed in XSS attack detection, such as data cleaning, feature extraction, feature selection, dimensionality reduction, and balancing datasets. Based on our review analysis results, these preprocessing techniques were categorized into different method types. We also investigated the machine and deep learning models utilized in the selected studies and identified the most used models for XSS attack detection. The review's analysis also examined the evaluation and validation approaches employed in the selected studies, including performance metrics, dataset types, data types, cross-validation, and data splitting percentages. The study identified the most frequently used preprocessing approaches, learning methods, performance metrics, dataset types, and data types in the selected studies. However, the trade-off between detection ability and efficiency remains a limitation, as most studies focus on improving detection accuracy rather than considering efficiency.

Overall, this systematic literature review provides a comprehensive and detailed overview of the recent advancements and trends in ML/DL-based XSS attack detection and can serve as a valuable reference for researchers and practitioners. The limitations and future research directions identified in this study can guide future research and development efforts towards improving of the effectiveness and efficiency of XSS attack detection methods.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declaration of Interest

The authors state that they have no known conflicting financial interests or personal ties that may have seemed to influence the work presented in this study.

References

Abaimov, S., & Bianchi, G. (2019). CODDLE: Code-Injection Detection with Deep Learning. *IEEE Access*, *7*, 128617–128627. https://doi.org/10.1109/ACCESS.2019.2939870

Akaishi, S., Uda, R., & IEEE. (2019). Classification of XSS Attacks by Machine Learning with Frequency of Appearance and Co-occurrence. In *2019 53rd Annual Conference on Information Sciences and Systems, CISS 2019*. IEEE. https://doi.org/10.1109/CISS.2019.8693047

Alazab, A., Khraisat, A., Alazab, M., & Singh, S. (2022). Detection of Obfuscated Malicious JavaScript Code. *Future Internet*, *14*(8), 217. https://doi.org/10.3390/fi14080217

Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2019). Code2Vec: Learning Distributed Representations of Code. *Proceedings of the ACM on Programming Languages*, *3*(POPL), 1–29. https://doi.org/10.1145/3290353

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. In *Journal of Big Data* (Vol. 8, Issue 1). Springer International Publishing. https://doi.org/10.1186/s40537-021-00444-8

Bai, S., Zico Kolter, J., & Koltun, V. (2018). *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. http://github.com/locuslab/TCN.

Barto, R. S. S. A. A. G. (2018). *Reinforcement Learning, second edition: An Introduction*. MIT press.

Carmen Torrano, G., Alejandro Pérez, V., & Gonzalo Álvarez, M. (2010). *HTTP DATASET CSIC 2010*. Information Security Institute of CSIC (Spanish Research National Council). https://www.tic.itefi.csic.es/dataset/

Chaiban, A., Sovilj, D., Soliman, H., Salmon, G., & Lin, X. (2022). Investigating the Influence of Feature Sources for Malicious Website Detection. *Applied Sciences (Switzerland)*, *12*(6). https://doi.org/10.3390/app12062806

Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers and Electrical Engineering*, *40*(1), 16–28. https://doi.org/10.1016/j.compeleceng.2013.11.024

Chaudhary, P., Gupta, B. B., Chang, X., Nedjah, N., & Chui, K. T. (2021). Enhancing big data security through integrating XSS scanner into fog nodes for SMEs gain. *Technological Forecasting and Social Change*, *168*, 120754. https://doi.org/https://doi.org/10.1016/j.techfore.2021.120754

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2011). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, *16*(Sept. 28), 321–357. https://doi.org/10.1613/jair.953

Chen, H.-C., Nshimiyimana, A., Damarjati, C., & Chang, P.-H. (2021). Detection and Prevention of Cross-site Scripting Attack with Combined Approaches. *2021 International Conference on Electronics, Information, and Communication (ICEIC)*, 1–4. https://doi.org/10.1109/ICEIC51217.2021.9369796

Chen, L., Tang, C., He, J., Zhao, H., Lan, X., & Li, T. (2022). XSS adversarial example attacks based on deep reinforcement learning. *Computers and Security*, *120*, 102831. https://doi.org/10.1016/j.cose.2022.102831

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13-17-Augu*, 785–794. https://doi.org/10.1145/2939672.2939785

Cheng, Z., Cui, B., & Fu, J. (2020). A novel web anomaly detection approach based on semantic structure. *Communications in Computer and Information Science*, *1298 CCIS*, 20–33. https://doi.org/10.1007/978-981-15-9031-3_2

Cheng, Z., Cui, B., Qi, T., Yang, W., & Fu, J. (2021). An Improved Feature Extraction Approach for Web Anomaly Detection Based on Semantic Structure. *Security and Communication Networks*, *2021*. https://doi.org/10.1155/2021/6661124

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 1–9. http://arxiv.org/abs/1412.3555

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/BF00994018

Cozamanis, A. (2019). *XSS Vectors Cheat Sheet · GitHub*. https://gist.github.com/kurobeats/9a613c9ab68914312cbb415134795b45

Cui, B., He, S., Shi, P., & Yao, X. (2018). Malicious URL detection with feature extraction based on machine learning. *International Journal of High Performance Computing and Networking*, *12*(2), 166–178. https://doi.org/10.1504/ijhpcn.2018.094367

Dong, Y., Wang, R., & He, J. (2019). Real-time network intrusion detection system based on deep learning. *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, *2019-Octob*, 1–4. https://doi.org/10.1109/ICSESS47205.2019.9040718

Duoergun. (2017). *GitHub*. https://github.com/duoergun0729/1book/tree/master/data

Fadel Waheed, W., & Alyasiri, H. (2023). Evolving trees for detecting android malware using evolutionary learning. *Int. J. Nonlinear Anal. Appl. In Press*, *14*(July 2022), 2008–6822. http://dx.doi.org/10.22075/ijnaa.2022.6874

Fang, Y., Huang, C., Xu, Y., & Li, Y. (2019). RLXSS: Optimizing XSS detection model to defend against adversarial attacks based on reinforcement learning. *Future Internet*, *11*(8). https://doi.org/10.3390/fi11080177

Fang, Y., Li, Y., Liu, L., & Huang, C. (2018). DeepXSS: Cross site scripting detection based on deep learning. *ACM International Conference Proceeding Series*, 47–51. https://doi.org/10.1145/3194452.3194469

Fang, Y., Xu, Y., Jia, P., & Huang, C. (2020). Providing Email Privacy by Preventing Webmail from Loading Malicious XSS Payloads. *Applied Sciences*, *10*(13), 4425. https://doi.org/10.3390/app10134425

Gao, C., Yan, J., Zhou, S., Varshney, P. K., & Liu, H. (2019). Long short-term memory-based deep recurrent neural networks for target tracking. *Information Sciences*, *502*, 279–296. https://doi.org/10.1016/j.ins.2019.06.039

Gao, C., Zhang, X., Han, M., & Liu, H. (2021). A review on cyber security named entity recognition. *Frontiers of Information Technology & Electronic Engineering*, *22*(9), 1153–1168. https://doi.org/10.1631/FITEE.2000286

Ge, L., & Moh, T. S. (2017). Improving text classification with word embedding. *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, *2018-Janua*, 1796–1805. https://doi.org/10.1109/BigData.2017.8258123

Geetha, R., & Thilagam, T. (2021). A Review on the Effectiveness of Machine Learning and Deep Learning Algorithms for Cyber Security. *Archives of Computational Methods in Engineering*, *28*(4), 2861–2879. https://doi.org/10.1007/s11831-020-09478-2

Ghaleb, F. A., Alsaedi, M., Saeed, F., Ahmad, J., & Alasli, M. (2022). Cyber Threat Intelligence-Based Malicious URL Detection Model Using Ensemble Learning. *Sensors*, *22*(9). https://doi.org/10.3390/s22093373

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139–144. https://doi.org/10.1145/3422622

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, *18*(5–6), 602–610. https://doi.org/10.1016/j.neunet.2005.06.042

Gupta, C., Singh, R. K., & Mohapatra, A. K. (2022). GeneMiner: A Classification Approach for Detection of XSS Attacks on Web Services. *Computational Intelligence and Neuroscience*, *2022*(1), 1–12. https://doi.org/10.1155/2022/3675821

Gupta, S., & Gupta, B. B. (2017). Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of Systems Assurance Engineering and Management*, *8*, 512–530. https://doi.org/10.1007/s13198-015-0376-0

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Hewamalage, H., Bergmeir, C., & Bandara, K. (2021). Recurrent Neural Networks for Time Series Forecasting: Current status and future directions. *International Journal of Forecasting*, *37*(1), 388–427. https://doi.org/10.1016/j.ijforecast.2020.06.008

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

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

Huang, Y., Li, T., Zhang, L., Li, B., & Liu, X. (2021). JSContana: Malicious JavaScript detection using adaptable context analysis and key feature extraction. *Computers and Security*, *104*, 102218. https://doi.org/10.1016/j.cose.2021.102218

Hydara, I., Sultan, A. B. M., Zulzalil, H., & Admodisastro, N. (2015). Current state of research on cross-site scripting (XSS) - A systematic literature review. *Information and Software Technology*, *58*, 170–186. https://doi.org/10.1016/j.infsof.2014.07.010

HynekPetrak. (2017). *Javascript Malware Collection*. https://github.com/HynekPetrak/javascript-malware-collection

I. Sharafaldin, A. Habibi Lashkari, & A. A. Ghorbani. (2017). *CIC-IDS 2017 Datasets Canadian Institute for Cybersecurity | UNB*. https://www.unb.ca/cic/datasets/ids-2017.html

ISCX-URL. (2016). *URL Datasets*. Canadian Institute for Cybersecurity (UNB). https://www.unb.ca/cic/datasets/url-2016.html

İsmail Taşdelen. (2021). *XSS Payload Box*. https://github.com/payloadbox/xss-payload-list/

John-Otumu, A., Imhanlahimi, A., & Akpe, R. (2018). Cross Site Scripting Attacks in Web-Based Applications: A Critical Review on Detection and Prevention Techniques. *Journal of Advances in Science and Engineering*, *1*(January), 25–35.

Kareem Thajeel, I., Samsudin, K., Jahari Hashim, S., & Hashim, F. (2023). Dynamic feature selection model for adaptive cross site scripting attack detection using developed multi-agent deep Q learning model. *Journal of King Saud University - Computer and Information Sciences*, *xxxx*. https://doi.org/10.1016/j.jksuci.2023.01.012

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

Kaur, J., Garg, U., & Bathla, G. (2023). Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review. *Artificial Intelligence Review*. https://doi.org/10.1007/s10462-023-10433-3

Kim, Y. (2014). Convolutional neural networks for sentence classification. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 1746–1751. https://doi.org/10.3115/v1/d14-1181

Kitchenham, B. A., Budgen, D., & Brereton, P. (2015). Evidence-Based Software Engineering and Systematic Reviews. In *Chapman & Hall/CRC*. Chapman and Hall/CRC. https://doi.org/10.1201/b19467

Kuhrmann, M., Fernández, D. M., & Daneva, M. (2017). On the pragmatic design of literature studies in software engineering: an experience-based guideline. *Empirical Software Engineering*, *22*(6), 2852–2891. https://doi.org/10.1007/s10664-016-9492-y

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

Le, Q., & Mikolov, T. (2014). Distributed Representations of Sentences and Documents. In E. P. Xing & T. Jebara (Eds.), *Proceedings of the 31st International Conference on Machine Learning* (Vol. 32, Issue 2, pp. 1188–1196). PMLR. https://proceedings.mlr.press/v32/le14.html

Lee, S., Wi, S., & Son, S. (2022). Link: Black-Box Detection of Cross-Site Scripting Vulnerabilities Using Reinforcement Learning. *WWW 2022 - Proceedings of the ACM Web Conference 2022*, *April*, 743–754. https://doi.org/10.1145/3485447.3512234

Lei, L., Chen, M., He, C., & Li, D. (2020). XSS Detection Technology Based on LSTM-Attention. *2020 5th International Conference on Control, Robotics and Cybernetics (CRC)*, 175–180. https://doi.org/10.1109/CRC51253.2020.9253484

Li, X., Ma, W., Zhou, Z., & Xu, C. (2020). XSS Attack Detection Model Based on Semi-supervised Learning Algorithm with Weighted Neighbor Purity. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 12338 LNCS* (pp. 198–213). Springer International Publishing. https://doi.org/10.1007/978-3-030-61746-2_15

Liu, M., Zhang, B. Y., Chen, W. B., & Zhang, X. L. (2019). A Survey of Exploitation and Detection Methods of XSS Vulnerabilities. *IEEE ACCESS*, *7*, 182004–182016. https://doi.org/10.1109/ACCESS.2019.2960449

Liu, Z., Fang, Y., Huang, C., & Han, J. (2021). GraphXSS : An Efficient XSS Payload Detection Approach Based on Graph Convolutional Network. *Computers & Security*, 102597. https://doi.org/10.1016/j.cose.2021.102597

Liu, Z., Fang, Y., Huang, C., & Xu, Y. (2022). GAXSS: Effective Payload Generation Method to Detect XSS Vulnerabilities Based on Genetic Algorithm. *Security and Communication Networks*, *2022*. https://doi.org/10.1155/2022/2031924

Liu, Z., Fang, Y., Huang, C., & Xu, Y. (2023). MFXSS: An effective XSS vulnerability detection method in JavaScript based on multi-feature model. *Computers & Security*, *124*, 103015. https://doi.org/10.1016/j.cose.2022.103015

Lu, J., Wei, Z., Qin, Z., Chang, Y., & Zhang, S. (2022). Resolving Cross-Site Scripting Attacks through Fusion Verification and Machine Learning. *Mathematics*, *10*(20). https://doi.org/10.3390/math10203787

Malviya, V. K., Rai, S., & Gupta, A. (2021). Development of web browser prototype with embedded classification capability for mitigating Cross-Site Scripting attacks. *Applied Soft Computing*, *102*, 106873. https://doi.org/10.1016/j.asoc.2020.106873

Malviya, V., Rai, S., & Gupta, A. (2018). Development of a plugin based extensible feature extraction framework. *Proceedings of the ACM Symposium on Applied Computing*, 1840–1847. https://doi.org/10.1145/3167132.3167328

Marashdih, A. W., Zaaba, Z. F., Suwais, K., & Mohd, N. A. (2019). Web application security: An investigation on static analysis with other algorithms to detect cross site scripting. *Procedia Computer Science*, *161*, 1173–1181. https://doi.org/10.1016/j.procs.2019.11.230

Maurel, H., Vidal, S., & Rezk, T. (2022). Statically identifying XSS using deep learning. *Science of Computer Programming*, *219*, 102810. https://doi.org/10.1016/j.scico.2022.102810

Melicher, W., Fung, C., Bauer, L., & Jia, L. (2021). Towards a Lightweight, Hybrid Approach for Detecting DOM XSS Vulnerabilities with Machine Learning. *Proceedings of the Web Conference 2021*, 2684–2695. https://doi.org/10.1145/3442381.3450062

Mereani, F. A., & Howe, J. M. (2018). Detecting Cross-Site Scripting Attacks Using Machine Learning. In *Advances in Intelligent Systems and Computing* (Vol. 723). https://doi.org/10.1007/978-3-319-74690-6_20

Mienye, I. D., & Sun, Y. (2022). A Survey of Ensemble Learning: Concepts, Algorithms, Applications, and Prospects. *IEEE Access*, *10*(September), 99129–99149. https://doi.org/10.1109/ACCESS.2022.3207287

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, 1–12.

Mohammadi, S., & Namadchian, A. (2020). Anomaly-based Web Attack Detection: The Application of Deep Neural Network Seq2Seq With Attention Mechanism. *ISECURE-ISC INTERNATIONAL JOURNAL OF INFORMATION SECURITY*, *12*(1), 44–54. https://doi.org/10.22042/ISECURE.2020.199009.479

Mokbal, F. M. M., Dan, W., Imran, A., Lin, J. C., Akhtar, F., & Wang, X. X. (2019). MLPXSS: An Integrated XSS-Based Attack Detection Scheme in Web Applications Using Multilayer Perceptron Technique. *IEEE ACCESS*, *7*, 100567–100580. https://doi.org/10.1109/ACCESS.2019.2927417

Mokbal, F. M. M., Dan, W., Xiaoxi, W., Wenbin, Z., & Lihua, F. (2021). XGBXSS: An Extreme Gradient Boosting Detection Framework for Cross-Site Scripting Attacks Based on Hybrid Feature Selection Approach and Parameters Optimization. *Journal of Information Security and Applications*, *58*(March), 102813. https://doi.org/10.1016/j.jisa.2021.102813

Mokbal, F. M. M., Wang, D., Wang, X. X., & Fu, L. H. (2020). Data augmentation-based conditional Wasserstein generative adversarial network-gradient penalty for XSS attack detection system. *PEERJ COMPUTER SCIENCE*, *6*, 1–20. https://doi.org/10.7717/peerj-cs.328

Nagarjun, P., & Ahamad, S. S. (2020). Ensemble methods to detect XSS attacks. *International Journal of Advanced Computer Science and Applications*, *11*(5), 695–700. https://doi.org/10.14569/IJACSA.2020.0110585

Nagarjun, P. M. D., & Ahamad, S. S. (2020). Cross-site Scripting Research: A Review. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*,

11(4), 626–632.

Ngoc, P. M., & Mimura, M. (2021). Oversampling for Detection of Malicious JavaScript in Realistic Environment. *Lecture Notes in Networks and Systems*, *159 LNNS*, 176–187. https://doi.org/10.1007/978-3-030-61108-8_17

Niu, Q., & Li, X. (2020). A High-performance Web Attack Detection Method based on CNN-GRU Model. *Proceedings of 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference, ITNEC 2020*, *Itnec*, 804–808. https://doi.org/10.1109/ITNEC48623.2020.9085028

Nti, I. K., Adekoya, A. F., & Weyori, B. A. (2020). A comprehensive evaluation of ensemble learning for stock-market prediction. *Journal of Big Data*, *7*(1). https://doi.org/10.1186/s40537-020-00299-5

OWASP Top Ten Web Application Security Risks | OWASP. (n.d.). Retrieved February 23, 2021, from https://owasp.org/www-project-top-ten/

Pan, H., Fang, Y., Huang, C., Guo, W., & Wan, X. (2022). GCNXSS: An Attack Detection Approach for Cross-Site Scripting Based on Graph Convolutional Networks. *KSII Transactions on Internet and Information Systems*, *16*(12), 4008–4023. https://doi.org/10.3837/tiis.2022.12.013

Pan, Y., Sun, F., Teng, Z., White, J., Schmidt, D. C., Staples, J., & Krause, L. (2019). Detecting web attacks with end-to-end deep learning. *Journal of Internet Services and Applications*, *10*(1). https://doi.org/10.1186/s13174-019-0115-x

Pan, Z., Yu, W., Yi, X., Khan, A., Yuan, F., & Zheng, Y. (2019). Recent Progress on Generative Adversarial Networks (GANs): A Survey. *IEEE Access*, *7*, 36322–36333. https://doi.org/10.1109/ACCESS.2019.2905015

Panigrahi, R., Borah, S., Pramanik, M., Bhoi, A. K., Barsocchi, P., Nayak, S. R., & Alnumay, W. (2022). Intrusion detection in cyber–physical environment using hybrid Naïve Bayes—Decision table and multi-objective evolutionary feature selection. *Computer Communications*, *188*(February), 133–144. https://doi.org/10.1016/j.comcom.2022.03.009

PhishTank. (n.d.). *Malicious URL Phishing*. Retrieved March 28, 2023, from https://phishtank.org/

Phung, N. M., & Mimura, M. (2021). Detection of malicious javascript on an imbalanced dataset. *Internet of Things*, *13*, 100357. https://doi.org/10.1016/j.iot.2021.100357

Qin, Z.-Q., Ma, X.-K., & Wang, Y.-J. (2018). Attentional Payload Anomaly Detector for Web Applications. In L. Cheng, A. C. S. Leung, & S. Ozawa (Eds.), *Neural Information Processing* (pp. 588–599). Springer International Publishing.

R. Hansen, & J. Manico. (2018). *XSS Filter Evasion Cheat Sheet | OWASP*. https://owasp.org/www-community/xss-filter-evasion-cheatsheet

Ray, S. (2019). A Quick Review of Machine Learning Algorithms. *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 35–39. https://doi.org/10.1109/COMITCon.2019.8862451

Refaeilzadeh, P., Tang, L., & Liu, H. (2009). *Cross-Validation BT - Encyclopedia of Database Systems*

(L. LIU & M. T. ÖZSU (Eds.); pp. 532–538). Springer US. https://doi.org/10.1007/978-0-387-39940-9_565

Ribeiro, M. H. D. M., & dos Santos Coelho, L. (2020). Ensemble approach based on bagging, boosting and stacking for short-term prediction in agribusiness time series. *Applied Soft Computing Journal*, *86*, 105837. https://doi.org/10.1016/j.asoc.2019.105837

Rodríguez, G. E., Torres, J. G., Flores, P., & Benavides, D. E. (2020). Cross-site scripting (XSS) attacks and mitigation: A survey. *Computer Networks*, *166*. https://doi.org/10.1016/j.comnet.2019.106960

Rozi, M. F., Ozawa, S., Ban, T., Kim, S., Takahashi, T., & Inoue, D. (2022). Understanding the Influence of AST-JS for Improving Malicious Webpage Detection. *Applied Sciences (Switzerland)*, *12*(24). https://doi.org/10.3390/app122412916

Sagi, O., & Rokach, L. (2018). Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *8*(4), 1–18. https://doi.org/10.1002/widm.1249

Samek, W., Montavon, G., Lapuschkin, S., Anders, C. J., & Müller, K. R. (2021). Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications. *Proceedings of the IEEE*, *109*(3), 247–278. https://doi.org/10.1109/JPROC.2021.3060483

Sarmah, U., Bhattacharyya, D. K., & Kalita, J. K. (2018). A survey of detection methods for XSS attacks. *Journal of Network and Computer Applications*, *118*(April), 113–143. https://doi.org/10.1016/j.jnca.2018.06.004

Sarmah, U., Bhattacharyya, D. K., & Kalita, J. K. (2020). XSSD: A Cross-site Scripting Attack Dataset and its Evaluation. *2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP)*, 21–30. https://doi.org/10.1109/ISEA-ISAP49340.2020.234995

SHAH, S. S. H. (2020). *Cross site scripting XSS dataset for Deep learning*. Kaggle. https://www.kaggle.com/datasets/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning

Shahid, W. Bin, Aslam, B., Abbas, H., Khalid, S. Bin, & Afzal, H. (2022). An enhanced deep learning based framework for web attacks detection, mitigation and attacker profiling. *Journal of Network and Computer Applications*, *198*(November 2021), 103270. https://doi.org/10.1016/j.jnca.2021.103270

Singh, A., Sharma, S., & Singh, J. (2021). Nature-inspired algorithms for Wireless Sensor Networks: A comprehensive survey. *Computer Science Review*, *39*, 100342. https://doi.org/10.1016/j.cosrev.2020.100342

Song, X. Y., Chen, C., Cui, B. J., & Fu, J. S. (2020). Malicious JavaScript Detection Based on Bidirectional LSTM Model. *APPLIED SCIENCES-BASEL*, *10*(10). https://doi.org/10.3390/app10103440

Stency, V. S., & Mohanasundaram, N. (2021). A Study on XSS Attacks: Intelligent Detection Methods. *Journal of Physics: Conference Series*, *1767*(1), 012047. https://doi.org/10.1088/1742-6596/1767/1/012047

Stiawan, D., Bardadi, A., Afifah, N., Melinda, L., Heryanto, A., Wanda Septian, T., Yazid Idris, M.,

Much Ibnu Subroto, I., Lukman, & Budiarto, R. (2023). An Improved LSTM-PCA Ensemble Classifier for SQL Injection and XSS Attack Detection. *Computer Systems Science and Engineering*, *46*(2), 1759–1774. https://doi.org/10.32604/csse.2023.034047

Stiawan, D., Wahyudi, D., Heryanto, A., Samsuryadi, Idris, M. Y., Muchtar, F., Alzahrani, M. A., & Budiarto, R. (2019). TCP FIN flood attack pattern recognition on Internet of Things with rule based signature analysis. *International Journal of Online and Biomedical Engineering*, *15*(7), 124–139. https://doi.org/10.3991/ijoe.v15i07.9848

Suleman, M. T., & Awan, S. M. (2019). Optimization of URL-Based Phishing Websites Detection through Genetic Algorithms. *Automatic Control and Computer Sciences*, *53*(4), 333–341. https://doi.org/10.3103/S0146411619040102

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2014). Intriguing properties of neural networks. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 1–10.

Tahmasebi, N., & Risse, T. (2011). Data Mining. In *Mining of Massive Datasets: Vol. 10450 LNCS* (pp. 1–17). Cambridge University Press. https://doi.org/10.1017/CBO9781139058452.002

Tama, B. A., Nkenyereye, L., Islam, S. M. R., & Kwak, K.-S. (2020). An Enhanced Anomaly Detection in Web Traffic Using a Stack of Classifier Ensemble. *IEEE Access*, *8*, 24120–24134. https://doi.org/10.1109/ACCESS.2020.2969428

Tariq, I., Sindhu, M. A. M. A. M. A., Abbasi, R. A. R. A., Khattak, A. S. A. S. A. S. A. S., Maqbool, O., & Siddiqui, G. F. G. F. G. F. (2021). Resolving cross-site scripting attacks through genetic algorithm and reinforcement learning. *Expert Systems with Applications*, *168*(August 2020), 114386. https://doi.org/10.1016/j.eswa.2020.114386

Tekerek, A. (2021). A novel architecture for web-based attack detection using convolutional neural network. *Computers and Security*, *100*. https://doi.org/10.1016/j.cose.2020.102096

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P. A. (2010). Stacked denoising autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research*, *11*, 3371–3408.

Vluymans, S. (2019). Learning from Imbalanced Data. In *Studies in Computational Intelligence* (Vol. 807, pp. 81–110). https://doi.org/10.1007/978-3-030-04663-7_4

VX Vault. (2017). *Malicious URL* . http://vxvault.net/ViriList.php

Wang, Q., Yang, H., Wu, G., Choo, K. R., Zhang, Z., Miao, G., & Ren, Y. (2022). Black-box adversarial attacks on XSS attack detection model. *Computers & Security*, *113*, 102554. https://doi.org/https://doi.org/10.1016/j.cose.2021.102554

Wang, R., Xu, G., Zeng, X., Li, X., & Feng, Z. (2018). TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting. *Journal of Parallel and Distributed Computing*, *118*, 100–106. https://doi.org/10.1016/j.jpdc.2017.07.006

Wang, X., Wang, H., & Wu, D. (2022). Dynamic feature weighting for data streams with distribution-based log-likelihood divergence. *Engineering Applications of Artificial Intelligence*, *107*(October

2021), 104509. https://doi.org/10.1016/j.engappai.2021.104509

Wooyun. (2020). *Wooyun-Email-XSS-Dataset*. GitHub. https://github.com/WhiteRabbitc/Wooyun-Email-XSS-Dataset/tree/master/malious-sample

XSSed Archive. (2015). *XSSed | Cross Site Scripting (XSS) attacks information and archive*. http://xssed.com/

Xu, H., Kotov, A., Dong, M., Carcone, A. I., Zhu, D., & Naar-King, S. (2016). Text classification with topic-based word embedding and Convolutional Neural Networks. *ACM-BCB 2016 - 7th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, 88–97. https://doi.org/10.1145/2975167.2975176

Yan, H., Feng, L., Yu, Y., Liao, W., Feng, L., Zhang, J., Liu, D., Zou, Y., Liu, C., Qu, L., & Zhang, X. (2022). Cross-site scripting attack detection based on a modified convolution neural network. *Frontiers in Computational Neuroscience*, *16*(August), 1–13. https://doi.org/10.3389/fncom.2022.981739

Yan, R., Xiao, X., Hu, G., Peng, S., & Jiang, Y. (2018). New deep learning method to detect code injection attacks on hybrid applications. *Journal of Systems and Software*, *137*, 67–77. https://doi.org/10.1016/j.jss.2017.11.001

Yang, J., Zhou, M., & Cui, B. (2020). MLAB-BiLSTM: Online Web Attack Detection Via Attention-Based Deep Neural Networks. *Communications in Computer and Information Science*, *1268 CCIS*, 482–492. https://doi.org/10.1007/978-981-15-9129-7_33

Yang, W., Zuo, W., & Cui, B. (2019). Detecting Malicious URLs via a Keyword-Based Convolutional Gated-Recurrent-Unit Neural Network. *IEEE Access*, *7*, 29891–29900. https://doi.org/10.1109/ACCESS.2019.2895751

Yong Fang, Yang Li, Liang Liu, & Cheng Huang. (2018). *GitHub - das-lab/deep-xss: deep-xss*. https://github.com/das-lab/deep-xss

Yu, Y., Yan, H., Guan, H., & Zhou, H. (2018). *DeepHTTP: Semantics-Structure Model with Attention for Anomalous HTTP Traffic Detection and Pattern Mining*. http://arxiv.org/abs/1810.12751

Yuan, X., He, P., Zhu, Q., & Li, X. (2019). Adversarial Examples: Attacks and Defenses for Deep Learning. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(9), 2805–2824. https://doi.org/10.1109/TNNLS.2018.2886017

Zhang, G., Nie, Y., & Wang, X. (2019). CNNPayl: An intrusion detection system of cross-site script detection. *ACM International Conference Proceeding Series*, *Part F1481*, 477–483. https://doi.org/10.1145/3318299.3318302

Zhang, X. Q., Zhou, Y., Pei, S. W., Zhuge, J. J., & Chen, J. H. (2020). Adversarial Examples Detection for XSS Attacks Based on Generative Adversarial Networks. *IEEE ACCESS*, *8*, 10989–10996. https://doi.org/10.1109/ACCESS.2020.2965184

Zheng, W., & Yin, L. (2022). Characterization inference based on joint-optimization of multi-layer semantics and deep fusion matching network. *PeerJ Computer Science*, *8*, 1–22. https://doi.org/10.7717/PEERJ-CS.908

Zhou, Y., & Wang, P. (2019). An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence. *Computers and Security*, *82*, 261–269. https://doi.org/10.1016/j.cose.2018.12.016

Declaration of Interest

The authors state that they have no known conflicting financial interests or personal ties that may have seemed to affect the work presented in this study.