HOSTED BY

ELSEVIER

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Dynamic feature selection model for adaptive cross site scripting attack detection using developed multi-agent deep Q learning model

Isam Kareem 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

ARTICLE INFO

Article history:
Received 5 November 2022
Revised 16 January 2023
Accepted 17 January 2023
Available online xxxx

Keywords:
Cross-Site Scripting (XSS) Attack
Web Application Security
Feature Drift
Dynamic Feature Selection
Multi-agent Reinforcement Learning

ABSTRACT

Web applications' popularity has raised attention in various service domains, which increased the concern about cyber-attacks. One of these most serious and frequent web application attacks is a Cross-site scripting attack (XSS). It causes grievous harm to victims. Existing security methods against XSS fail due to the evolving nature of XSS attacks. One evolving aspect of XSS attacks is feature drift which changes the feature relevancy and causes degradation in the performance. Unfortunately, dynamic awareness of drift occurrence is missing. Thus, this study attempts to fill the gap by proposing a feature drift-aware algorithm for detecting the evolved XSS attacks. The proposed approach is a dynamic feature selection based on a deep Q-network multi-agent feature selection (DQN-MAFS) framework. Each agent is associated with one feature and is responsible for selecting or deselecting its feature. DQN-MAFS provides a sub-model for reward distribution over agents, which is named as fair agent reward distribution based dynamic feature selection FARD-DFS. This framework is capable of supporting real-time, dynamic updates and adjustment of embedded knowledge as long as new labelled data arrives. DQN-MAFS has been evaluated using four real XSS attack datasets with various feature length sizes. The evaluation process was conducted and compared with state-of-the-art works. The obtained results show the superiority of our FARD-DFS over the benchmarks in terms of the majority of metrics. The improvement percentages of the mean accuracy and F1-measure ranged from 1.01% to 12.1% and from 0.55% to 6.88%, respectively, in comparison with the benchmarks. This approach can be deployed as an autonomous detection system without the need for any offline retraining process of the model to detect the evolved XSS attack.

1. Introduction

In modern days, people are surrounded by billions of web application pages in many distinct domains; with just a single click by the user, enormous amounts of data streams are produced at extremely high-speed rates. Most of these web applications are high-frequency use applications that impose fast-evolving data to respond immediately and dynamically (X. Wang et al., 2022). In this case, the data is processed in real-time and the browser environment is changed due to the client interactions. On the other side, the adoption of Industry 4.0 and 5.0 has been expedited by recent developments in information and communication platforms, such as big data, cloud, artificial intelligence, Internet-of-things (IoT), and nanotechnology. Industry sectors confront new concerns due to cybersecurity threats caused mainly by these technological improvements. The frequency of cyberattacks directed against the industry sector has risen dramatically in recent years (S. Kumar & Mallipeddi, 2022). At the same time, cyber threats are evolving to conduct attacks with high severity levels (Sun et al., 2019). One of these evolving attacks is cross-site scripting (XSS) (Sarmah et al., 2020; Tariq et al., 2021; Zhou & Wang, 2019), as the industry 4.0 and 5.0 rely heavily on web-based interfaces to provide remote access and control of industrial processes. An XSS attack is a special type of injection attack that occurs on the application level. The attacker can conduct the XSS attack by exploiting the vulnerabilities that may be found in Web applications due to the use of improper or dangerous scripts by developers (Sarmah et al., 2018). An XSS attack is used to threaten various domains such as industries, individual privacy, medical, economics, and even the military (Rodríguez et al., 2020). In the

* Corresponding authors.

E-mail addresses: isamkthajeel@gmail.com (I. Kareem Thajeel), khairulmizam@upm.edu.my (K. Samsudin).

Peer review under responsibility of King Saud University.

ELSEVIER Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2023.01.012
1319-1578/© 2023 The Author(s). Published by Elsevier B.V. on behalf of King Saud University.

Please cite this article as: I. Kareem Thajeel, K. Samsudin, S. Jahari Hashim et al., 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 Scienceshttps://doi.org/10.1016/j.jksuci.2023.01.012

context of industry 4.0 and 5.0, there has been a significant increase in the use of internet-of-things (IoT) sensors with supervisory-control and data-acquisition (SCADA) systems for monitoring and controlling the industrial processes by allowing remote access. Though SCADA has improved control and manageability, it has also subjected such systems to cyberattacks. XSS is one of these threats, according to an investigation conducted by (Upadhyay et al., 2020) in the analysis of Omega2, a tiny Linux server from OnionTM that is employed to construct different SCADA-IoT systems. Typically, the attacker uses the same steps to perform the XSS regardless of which domain is targeting, but the only difference is the type of exposed information. At the initial step, the attacker looks for and identifies potential vulnerabilities in the web application (Z. Liu et al., 2023). A malicious script is created and injected into a susceptible Web application's trusted context. The URL link of this web application is given to the victim via various media. When the victim is lured to click on the received link, the browser executes the malicious script that masquerades as part of the legitimate code because of the browser's inability to distinguish between malicious and normal scripts. As a result, the users of vulnerable applications fall prey to attackers who obtain credentials information. In industry domain, the attacker could then use these exploited credentials to gain unauthorized access to the SCADA web-based interface systems, control production systems, disrupt the production process, and steal sensitive production data. In general, the total XSS attacks has sharply risen over the years.[1] Currently, the XSS has the third-highest total count with 23,363 attacks from its discovery time to September 2022, making approximately 12.7 % of the total reported vulnerabilities. Furthermore, XSS has been included in every Open Web Application Security-Project (OWASP) top-10 list to present.

Various classes of methods were used to counter XSS. Initially, with the emergence of XSS attacks to the public, specialists relied on traditional methods to detect the XSS attacks on the client-side, server-side, or pair sides. These approaches can be broadly classified into three kinds: static, dynamic, and hybrid analysis (Mereani & Howe, 2018; Rodríguez et al., 2020). The static analysis method focuses on reviewing the source code without running it, while the dynamic analysis approach focuses on analysing the behaviour of the script during execution with some input. Finally, Hybrid analysis combines static and dynamic analysis to mitigate the disadvantages of both techniques. Static, dynamic, and hybrid are regarded as classical XSS defence methods. Such classical methods might provide some promises that such XSS vulnerabilities will not occur. However, they are also criticized for being slow, unable to provide an efficient result due to its high false positive rates, complex, costly in terms of both resource-consuming and time-consuming, and requiring high-security domain knowledge (M. Liu et al., 2019; R. Wang et al., 2018).

Current researchers attempt to leverage artificial intelligence (AI) principles to add intelligence to boost detection, as seen by the widespread usage of machine learning/deep learning (ML/DL) techniques in detecting cyber-attacks. On the other hand, traditional machine-learning techniques have a low detection rate and are incapable of detecting minor mutants of current malicious attacks, such as zero-day attacks (Mokbal et al., 2019; Sahoo et al., 2019). Moreover, the evolving nature of attacks requires an ML/DL-based approach equipped with dynamic awareness. More specifically, the monitoring of drift occurrence and adapting the model accordingly. One special type of drift that researchers in web security applications have ignored is feature drift, which indicates the dynamic or temporal change in the relevancy of features. Such change leads to a model trained with part of non-relevant features, and consequently, a degradation in the accuracy occurs (Barddal et al., 2017; Sato et al., 2022). To handle such issues, researchers relied on traditional feature selection methods to select the most relevant features. The traditional feature selection methods have a static nature and are based on batch learning which assumes that the set of relevant features does not change over time and needs multiple passes over data. However, in many applications, and XSS is one of them, the features might be subject to change in their relevancy with respect to time due to the evolving nature of attacks. The evolved XSS attack can be defined as confusing and aggressive innovative XSS attacks that are generated with different combinations of features from the benign or malicious existing XSS records (Gupta et al., 2022). Usually, such combinations are created by using escaping strategies such as encoding, obfuscation techniques, replacing the alphabet style lowercase/uppercase, removing, or substituting the JavaScript and HTML attributes/functions/events/tags/symbols/expression, adding or removing the blank spaces, etc. The various combinations are highlighted by security experts (OWASP, XSS cheat sheet) (Jim Manico, 2018) and researchers' validation analysis on different repositories (Stock et al., 2017).

Feature drift has emerged only recently, and a limited number of techniques have been developed so far (Barddal et al., 2017). Dynamic feature selection (DFS) is an emerging sub-topic in machine learning that has been used recently to capture feature drift (Barddal et al., 2016; Ferone & Maratea, 2021). It involves selecting an optimal subset of features based on their relevancy and filtering out non-relevant features with respect to a set of records. DFS can be applied to both stream data and non-stream data. However, the DFS is more convenient for the stream and sequential nature of data due to the drift that indicates changing the distribution that represents the source of the data. It is essential to distinguish between terms, namely, feature drift which indicates relevancy change with respect to time and concept drift which indicates data distribution change with respect to time (Barddal et al., 2017). The former is the focus of this article within the application of XSS detection.

Hence, this article aims at handling this gap by proposing DFS based on a deep Q-network multi-agent feature selection (DQN-MAFS) framework. The DQN-MAFS selects the relevant features dynamically to update the detection model's knowledge over time. The suggested methods are evaluated using four real XSS datasets and compared with the state-of-the-art works. This article presents several contributions. They can be stated as follows:

- It is the first article that provides a dynamic feature selection in the domain of XSS detection by considering incremental learning and enabling knowledge updates for both feature selection and classification at the same time. This provides a feature drift aware XSS detection algorithm.
- The article proposes a novel framework for multi-agent dynamic feature selection using RL by including several components. The framework is named as deep Q-network multi agent feature selection DQN-MAFS. Furthermore, it provides a sub-model for reward distribution which is named as fair agent reward distribution based dynamic feature selection FARD-DFS.
- It provides four reward distribution methods for FARD-DFS, namely, accumulative contribution (ACC), alternative contribution (ALC) and impurity based (IM), and one action at one time (OA-OT). Furthermore, it allows simultaneous action change to the agents, which enables fast exploring of the solution space and increases the efficiency of the learning.

The remaining article is organized as follows. Section 2 summarizes the main contributions of the article. Section 3

[1] Vulnerability distribution of cve security vulnerabilities by types (cvedetails.com).

reviews the literature regarding ML/DL-based XSS detection approaches and reinforcement learning-based dynamic feature selection works. Section 4 describes the proposed DQN-MAFS methodology in detail. Section 5 demonstrates the experimental design, numerical, time series visualization, and memory reduction results. Section 6 presents this work's conclusion and suggests potential future aspects that can carry out further work.

2. Background

$R_t(F) : X \to Y$ is relevancy function that is related to time t where:

F represents the set of features that we want to check their relevancy.

X Denotes the candidate combination of features.

Y denotes a decision variable that has value of 0 or 1 where 0 indicates to non-relevancy and one indicates to relevancy relationship.

The feature drift happens if and only if

$$\exists\ t_1, t_2 \in [0, t], F \in X,\ t_1 \neq t_2, \text{ where } R_{t_1}(F) \neq R_{t_2}(F)$$

When a feature drift happens at some moment in the interval $[0, t]$, there are four main types of feature drift (Ferone & Maratea, 2021). We present them in Fig. 1.

- **Sudden drift:** It occurs when a specific feature throughout a specific number of time periods drifts from being relevant to irrelevant for classification and vice versa. The feature should either be removed or added to the optimal subset of features based on their relevancy changes. The feature relevancy changes remain in that state for a certain number of consecutive time periods.
- **Gradual drift:** It happens When several successive time periods are considered, and the frequency of a feature being considered relevant or irrelevant changes with time. In this case, a feature initially sporadically emerges and then vanishes from the subset of relevant features before stabilising.
- **Incremental drift:** It happens when the feature relevance within several successive time periods is represented by a serial number in the range of [0, 1] that grows or declines incrementally.
- **Reoccurring drift:** It occurs when a feature loses its relevance suddenly during a specific period and becomes significant again in subsequent periods, and vice versa, with no discernible pattern. If the changeover is rapid, it can turn into a reoccurring shift.

Evolving XSS attacks might encounter any of these types or even a combination of them. For example, sudden drift may occur with Zero attacks. In contrast, the gradual or incremental or reoccurring feature drift can be seen when the attacker uses the escape strategies to evolve their attack and generate a new combination attack. Some attack vectors are produced by the evolved strategies used by attackers, such as the mutation of some HTML events and properties, including innerHTML, outerHTML, etc. These new classes of XSS can be considered a sudden drift caused by bypassing almost all the XSS filters on the server-side and client-side (Heiderich et al., 2013).

3. Related work

This section presents the literature survey. It is decomposed into two sub-sections: the first one is about XSS detection methods using ML/DL in the literature. The second one is about dynamic feature selection using RL.

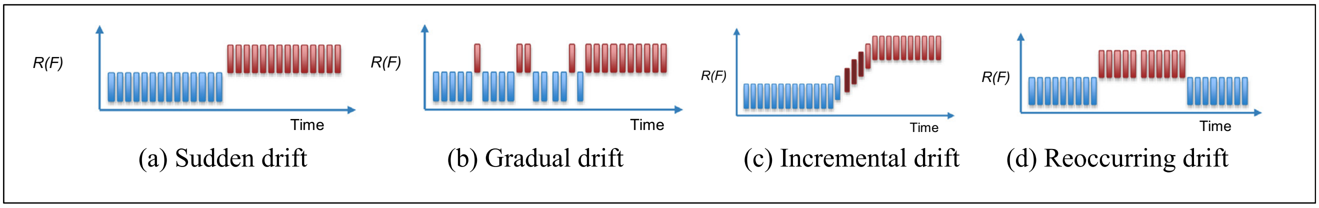
3.1. ML/DL-based XSS detection

The literature contains numerous ML/DL-based methods and algorithms for detecting XSS attacks:

Some methods were developed based on traditional classifiers. Artificial Neural Network (ANN) and Support Vector Machine (SVM) have been used by researchers. In the work of (Alazab et al., 2022), an automatic IDS is suggested by using machine learning for JavaScript attack detection. Other researchers have used a set of traditional machine learning classifiers. In the work of (Malviya et al., 2021), various features types of XSS were used. The classification includes several machine learning classifiers: Naïve-Bayes, SVM, Random Forest, and ADTree. In the work of (Mokbal et al., 2019), an ANN-based multilayer perceptron (MLP) scheme integrated with the preprocessing components is proposed for XSS attack.

Other researchers have used ensemble learning-based approaches. In the work of (Mokbal et al., 2021), a development of Extreme Gradient Boost (XGBoost) based XSS detection is proposed. The development is based on a new web-based XSS attack detection framework that uses an ensemble-learning technique. In addition, a combination of information gain (IG) with sequential backward selection (SBS) to select relevant subset features. In the work of (Mokbal et al., 2020), an integration of a conditional Wasserstein generative adversarial network with a gradient penalty was presented for XSS attack detection. In the work of (Zhou & Wang, 2019), the set of Bayesian networks (BNs) model is used to build the ensemble learning model. Their learning algorithm adopted scoring and searching techniques to evaluate nodes' importance based on their influences. A voting method is also applied here to ensemble the individual model to create the ensemble learner.

Another direction of ML/DL-based methods is the usage of reinforcement learning (RL). In the work of (Tariq et al., 2021), an integrated XSS detection approach based on three models was proposed, namely, A Genetic Algorithm (GA), Statistical Inference, and Reinforcement Learning (RL). First, the best chromosomes for detecting a vulnerability are compared. If differentiation cannot be achieved, fit chromosomes are compared to find the most differentiating chromosomes first. In this phase, a payload will most likely be classified as susceptible or non-vulnerable, but if the payload cannot be distinguished, the next phase of statistical inference will be performed. This RL is used to adapt to new attacks and is accomplished by altering, deleting, and adding chromosomes



(a) Sudden drift (b) Gradual drift (c) Incremental drift (d) Reoccurring drift

Fig. 1. Main types of feature drift.

inside the model. The Distinguisher, Declarator, and Model are the components that the RL module employs to improve performance. However, this work can be criticized for lacking structure for knowledge preservation, such as a neural network. In addition, it does not explicitly address the issues of feature drift that are caused by the XSS attack evolution.

The usage of deep neural networks (DNN) in general and convolutional neural networks (CNN) in particular was also found by other researchers. In another study by (Maurel et al., 2022), an experimental and comparative study was introduced by static approaches for XSS vulnerabilities detection with deep neural networks. The obtained results from several experiments that have been conducted with different settings of deep neural network hyper-parameters were compared. However, this method suffers from high false positive (FP) and false negative (FN) rates for both given methods. In the study of (J. Kumar et al., 2022), the CNN algorithm is used to detect XSS attacks. This method relayed completely on the CNN components in feature extraction on feature map layers and feature reduction on pooling layers. In the work of (Chaudhary et al., 2021), CNN has been used after two steps of data preparation, namely, decoding and contextual tagging. The algorithm has been deployed in Fog nodes connected with IoT network. In the work of (Abaimov & Bianchi, 2019), a model called CODDLE is proposed to detect the injection attacks such as XSS and SQL injection. In this model, a CNN algorithm is adopted. The extraction results are used as features for training the CNN. In the work of (Z. Liu et al., 2021), a graph convolutional network-based XSS payload detection model was proposed. The graph convolutional and residual networks were used to train the XSS detection model after preprocessing the sample using word2vec and constructing the processed data into a graph structure.

The usage of recurrent neural networks (RNN) in general and long short-term memory (LSTM) was also used by some approaches for effective feature learning. In the work of (Huang et al., 2021), JSContana consists of flexible text analysis based on dynamic word embeddings using word2vec and Bi-LSTM of both the segments (payloads) and the words (keywords) of feature vectors. Finally, the Text-CNN is used for classification. In the work of (Y. Fang et al., 2020), an email XSS detection model using Bidirectional-RNN algorithm and the Attention mechanism was proposed. Word2Vec model was used for pre-processing the email information and fed them for the classification model.

Many researchers refer to the Evolving aspect of XSS (Applebaum et al., 2021; Singh, 2020; Sun et al., 2019; Ye et al., 2021), but it has not been under the focus of researchers despite its high impact on the detection of XSS. Furthermore, some studies show that the current detection scanners and approaches have failed to detect the evolved XSS attacks. Automated scanners, namely, BruteXSS,[2] XSpear,[3] XSSer,[4] XSSmap,[5] w3af,[6] wapiti,[7] have shown low recall when they are evaluated with mutated XSS attacks using reinforcement learning (Caturano et al., 2021). Furthermore, many of the models for XSS attack detection have shown failure in detection attacks when they are challenged with adversarial attack generation. This shows the limitation of offline trained models or even models that lack knowledge updates or awareness to the evolving nature of XSS attacks. For example, the study of (Q. Wang et al., 2022) has shown that the escape rate of LSTM has increased from 75 % to 86 % when it is tested with adversarial attack generation. Similarly, the SVM escape rate has increased from 80 % to 86 % under adversarial attacks.

Overall, it is found that the majority of researchers have addressed the problem of XSS detection from the perspective of feature extraction, reduction, classification, and data generation for improving performance, as shown in Table 1. However, the problem was almost handled as offline learning and online testing without tackling the changeable nature of data nor enabling continuous knowledge updates whenever new labelled data arrived. Such an aspect is crucial to be leveraged for countering the drift and evolving behaviour that occurs in XSS attacks. According to (Alazab et al., 2022; Gupta et al., 2022), where attackers generate new malicious payloads, each with a different combination and unique enough, by arbitrarily inserting or modifying certain components of the current XSS scripts, such a new generated XSS payload can evade the detection tools. In this study, we called such attack as an evolved XSS attack. Hence, we focus in this article on developing dynamic feature selection for XSS to counter the feature drift caused by evolved XSS attacks. Furthermore, we select the reinforcement learning model as a framework for developing our dynamic feature selection because of its capability of supporting real-time, dynamic update, and adjustment of embedded knowledge as long as new labelled data arrived with a novel rewards distribution schema proposed for this purpose. The following sub-section provides a survey of the existing algorithms of RL-based dynamic feature selection.

3.2. Dynamic feature selection

Only a few methods in the literature are found that specifically seek to identify and respond to feature drifts problem (Barddal et al., 2019). The most effective method to handle feature drift is the DFS which provides the classification model with only the most relevant features and ignores the irrelevant features. The current methods were reviewed by (Barddal et al., 2017), and the authors found that most of these methods focused on applying implicit DFS for streaming learning, while less attention has been paid to explicit DFS. However, almost all the current methods that perform implicit DFS, such as decision tree learning, decision rule learning, combination, and randomness, have limitations in identifying the potential concept or feature drift since it presumes that the underlying distribution remains constant over time (X. Wang et al., 2022). Thus, this article focused on the emerging trend in the literature to solve the problem of feature drift using explicit DFS based on RL based models. This is due to the effectiveness of RL in handling dynamic changes and updating embedded knowledge corresponding to them inside the RL agent. Different types of agents were used in the literature for this purpose.

The usage of traditional Q-learning has been found in many studies. In the work of (Paniri et al., 2021), a feature selection method based on Ant Colony Optimization (ACO) was proposed. The temporal difference (TD) reinforcement learning algorithm was used to accomplish a heuristic learning approach for the ACO. Features represent the states (S) and selecting the unvisited features by each ant represents a set of actions. Reward signals are composed of two criteria when ants take action. In the work of (Xu et al., 2021), a method was introduced based on fitting a discriminant function using Q-learning. The agent will select the classification action while the function is approximated based on the feedback from the decision of the agent. However, this algorithm was built based on a linear discriminant function which limiting it to simple data.

A more advanced type of agent was found, like deep deterministic policy gradient (DDPG). In the work of (Cheng et al., 2021), the feature selection process is modelled as a Markov Decision Process (MDP), and formalized as an RL problem. They proposed a Deep

[2] https://github.com/shawarkhanethicalhacker/BruteXSS-1.

[3] https://github.com/hahwul/XSpear.

[4] https://xsser.03c8.net.

[5] https://awesomeopensource.com/project/secdec/xssmap.

[6] https://w3af.org/.

[7] https://wapiti-scanner.github.io.

Table 1
Literature Survey of the existing ML/DL-based XSS detection approaches.

Authors/year	Components	Feature selection	Feature drift	Classification	Online learning
(Alazab et al., 2022)	Statistical analysis and SVM classifier	√	×	SVM	×
(Maurel et al., 2022)	Word embedding representation components and DNN	–	×	Sigmoid	×
(J. Kumar et al., 2022)	CNN	√	×	Softmax	×
(Tariq et al., 2021)	RL, GA, and statistical inference	–	–	Distance based	×
(Huang et al., 2021)	Word2vec, Dynamic word embedding using Bi-LSTM	–	–	TextCNN	×
(Mokbal et al., 2019)	Feature extractor and MLP	–	–	MLP	×
(Mokbal et al., 2021)	Preprocessing components and XGBoost classifier	√	–	Multiple classifier	
(Z. Liu et al., 2021)	Graph building, Word2Vec, and GCN	×	×	GCN	×
(Y. Fang et al., 2018)	Decode, tokenizer, generalizer, word2vec and LSTM	×	×	Softmax	×
(Mokbal et al., 2020)	CGAN and WGAN-GP. And detection model based on XGBoost algorithm	×	×	XGBoost	×
(Abaimov & Bianchi, 2019)	Pairs coding and CNN	×	×	Softmax	×
(Zhou & Wang, 2019)	Ensemble learning, domain knowledge and threat intelligence	×	×	Bayesian networks	×
(Malviya et al., 2021)	Feature extractor and Classification model	×	×	SVM and other classifiers	×
(Chaudhary et al., 2021)	Data preparation and CNN	×	×	–	×
(Y. Fang et al., 2020)	Data preparation, bi-RNN and attention	×	×	RNN	×
(Yang et al., 2020)	Bi-LSTM With two levels of attention word and segment, and TextCNN	×	×	Softmax	×
Proposed approach	RL, DQN,	√	√	Multiple classifiers	√

Reinforcement Learning-based Feature Selector (DRLFS), as well as a dynamic randomness policy for controlling exploration and exploitation, and two search protocols, namely, fixed depth search and adaptive depth search.

Other researchers use deep Q-learning network-based agents (DQN) for DFS. In the work of (K. Liu et al., 2021), a multi-agent framework was proposed. Each agent is responsible for one feature, and the overall decisions of all agents provide the selected features. The agent's state is built based on descriptive statistics; the reward is a weighted average of three terms: relevancy, redundancy, and accuracy. The learning agent embeds DQN, which is trained using experience replay buffer sampling by using GMM. However, this framework did not include a unique mechanism for exploration and exploitation balance. The algorithm suffers from non-efficiency when the number of features increases due to the linear relationship between the features' size length and created DQN, which makes it limited in the application of XSS. In the work of (Fan et al., 2020), a navigation mechanism using interactive reinforcement learning (IRL) as an external trainer-agent interaction was proposed. The decision tree feedback (DTF) was used as learning feedback. Graph Convolutional Network (GCN) is adopted to learn state representation from both the graph and the tree at the same time. A customized reward assignment is set for agents depending on the relevance of decision tree features. In the work of (Z. Fang et al., 2019), A framework architecture of a Deep Q-learning based Feature Selection (DQFSA) was proposed for malware detection. The Q-learning agent is implemented with a sequential learning manner. In addition, the accuracy of the machine learning classifiers is used as a reward. In the work of (Li et al., 2021), a Deep Q-Network has been proposed. A single agent is represented by DQN, which takes the action of one feature at one time. The reward is formulated as a piecewise equation combined with four branches. In the work of (Wu et al., 2022), a framework based on Double DQN (DDQN) algorithm has been proposed for Android malware classification. The RNN is adopted as the decision network of DDQN to accept the unfixed input length. Word embedding is applied for feature representation to find the semantic relevance of features. In the work of (Fan et al., 2021), a framework for Group-based Interactive Reinforced Feature Selection (GIRFS) was suggested. Their framework balances single-agent RFS and multi-agent RFS. In this formulation, they first divide the given features into groups based on feature similarity. Then, each group creates agents, with each agent taking action on its corresponding group. Furthermore, they propose a hierarchical teacher-like trainer to provide external action advice to agents.

The reviewed methods are summarised in Table 2, from the perspectives of the elements of RL, namely, state, action, reward, agent type, and exploration vs exploitation balance. It is observed that non-of the existing methods have provided an algorithm for reward distribution to handle the multi-agent rewarding. The tricky part of multi-agent rewarding is that the agents' contribution to the overall performance is not equal (Gronauer & Diepold, 2021; Nguyen et al., 2020). This is because each agent's decision is weighted based on the importance of its feature in the overall performance; figuring out this importance is not a straightforward process. Hence, it is necessary to come up with an approach that accomplishes fair reward distribution based on the real contribution of each agent.

4. Methodology

This section illustrates the methodology of the proposed approach called DQN-based multi-agent features selection (DQN-MAFS) to detect the evolved XSS attacks in an incremental manner dynamically. Firstly, we formulated the handling of streaming XSS attacks as the dynamic feature selection problem. Following that, the significant components of the DQN-MAFS framework are introduced, as shown in the following sub-sections.

4.1. Problem formulation

Assuming a stream data set $D = (x_t, y_t)$ with partial labeling where:

t denotes the time and x_t denotes the feature vector $x_t \in \mathbb{R}^m$

$$y_t = \begin{cases} y_{target,t,} \ \textit{if label is available} \\ \nabla, \ \textit{if label is not available} \end{cases} \tag{1}$$

The percentage of labelling is $R_t \in [r_{min}, r_{max}]$ indicates the number of labelled samples over the total number of samples within a small time-interval T.

The goal is to build a prediction of the label of the data stream $f(x_t) = \tilde{y}_t$ where the error of the prediction is minimized or

Table 2
Literature Survey of the existing algorithms in RL based dynamic feature selection.

Article	Multi agent	State	Action	Reward	Reward distribution	Agent	Exploration Exploitation Balance	Classifier
(Cheng et al., 2021)	×	Encoding of the subset of selected features	Binary decision	Decline of Error	×	DDPG	Randomness policy	SVM
(K. Liu et al., 2021)	√	Descriptive statistics	Binary decision	Accuracy, redundancy, relevance	×	DQN	$\varepsilon-$ greeduy	RF, LASSO, DT, SVM, XGBoost
(Fan et al., 2020)	√	GCN	Binary decision	DT based personal rewarding and Accuracy from hybrid teaching	×	DQN	$\varepsilon-$ greeduy	Decision tree
(Z. Fang et al., 2019)	√	Selected feature	Select/deselect	Accuracy	×	DQN	$\varepsilon-$ greeduy	Set of classifiers
(Paniri et al., 2021)	√	Feature	Select/deselect	Cos similarity and correlation	×	Q-learning	Polynomial decay rate	KNN
(Xu et al., 2021)	×	Feature	Class	Fixed values for rewarding and punishment	×	Q-learning	$\varepsilon-$ greeduy	Linear discriminant function
(Fan et al., 2021)	√	Graph Convolu- tional Network (GCN)	Selection /de-selection	Accuracy	×	N/A	$\varepsilon-$ greeduy	Decision tree
(Li et al., 2021)	×	Binary	Selection /de-selection	Piecewise based on the accuracy	×	DQN	$\varepsilon-$ greeduy	KNN
(Wu et al., 2022)	×	Binary	sequential selection by RNN	Accuracy	×	DDQN	$\varepsilon-$ greeduy	Set of classifiers
Proposed approach	√	Binary	Binary decision	Developed (FARD-DFS)	√	DQN	Decay ε greedy	Set of classifiers

$e = \sum_{t=t_0}^{t_c} (\tilde{y}_t - y_{target,t,})$ is minimum where t_0 denotes the starting time and t_c denotes the current time.

The function f is a feature selector function. Its mission is to assign for each feature $i = 1, \cdots m$ a binary value $b_i \in \{0, 1\}$ where b_i denotes the decision of whether the feature is selected or not. This process is to minimize the error e.

Mathematically, we write the problem as an optimization problem according to the following formula:

$$\begin{aligned} b^* &= (b_{i,1}, b_{i,2}, b_{i,3}, \cdots b_{i,m}) = argmin(e) \\ &= argmin\left(\sum_{t=t_0}^{t_c} (\tilde{y}_t - y_{target,t,})^2\right) \\ &= argmin(\sum_{t=t_0}^{t_c} (f(x_t) - y_{target,t,})^2) \end{aligned} \tag{2}$$

This formulation provides a Markov Decision Process (MDP) based modelling of the feature selection system where the current decision of feature selection represents an action that is made based on the state that is originated according to the previous taken action and based on the obtained reward that is also calculated from our suggested reward mathematical formulation.

4.2. The general methodology

As shown in Fig. 2, the methodology consists of three main stages: (1) preparation of data streaming which is responsible for generating stream data from original XSS data (2) DQN-MAFS framework, which is responsible for dynamic feature selection (3) Detector which is responsible on classifying the records according to their feature's values.

4.3. Preparation of data streaming

The literature contains a wide range of use of the XSS data with ML/DL approaches. However, they all assume batch learning, and no sequential chunks were generated. In order to enable the incremental learning approach, we decompose the data into chunks as we provide them to the framework. The chunks are assumed based on the pre-given number of chunks. The time has not been used as a feature, but it helps in ordering the records (sequence order of the data). The data is presented as a stream of chunks to the method based on Eq. (1). Each chunk consists of a certain number of rows representing the records $x_1, x_2, x_3, \cdots, x_n$ and certain number of columns representing the features $f_1, f_2, f_3, \cdots, f_m$. The dimension of features is fixed, while the number of rows changes from one chunk to another according to the way the data is generated.

4.4. DQN-MAFS framework

The framework of DQN-MAFS is presented in Fig. 2. The framework supports multi-agent learning for feature selection, like (K. Liu et al., 2021) in general and different in details of various components such as state representation, reward schema, exploration, and exploitation policy. In our suggested DQN-MAFS, each agent is responsible for one feature, and it has one of two actions, namely, selection or de-selection. Each agent generates the state from the chunk and uses it for selecting or de-selecting the corresponding feature of the agent. The aim is to reduce the data to a more discriminative subset that improves prediction and efficiency. The agent contains a deep Q network trained based on a mini-batch sampled from an experience-replay buffer similar to (Mnih et al., 2013). The neural network is trained based on loss function using gradient descent.

This section provides the formulation of the elements of our DQN-MAFS. We define each of the environment, state, agent, action, and reward.

4.4.1. Environment

In general, the environment represents all entities outside the agent. In this research, the environment represents the data in terms of the number of features, their types, values, and their associated labels (ground truth) that provides the capability of learning the knowledge based on the calculated reward.

4.4.2. State

The state is represented by a binary vector that has a length equal to the number of features. The vector elements are zeros or ones based on the last generated decision of the agents for selecting or deselecting their features. The selection is encoded with one, while the de-selection is encoded by zero. Hence, the model maintains Markov Decision Process (MDP) or

$$p(S_t|S_{t-1}, S_{t-2}) = p(S_t|S_{t-1}) \tag{3}$$

4.4.3. Action

The action is associated with the agent decision of selecting or deselecting its feature. Assuming that we have a total m features, then the selection decision is a binary value ranging from 1, which is the decision to select the first features only to 2^m, which represents the decision of selecting the entire set of features. This is given by the Eq. (4).

$$a_t = \begin{cases} 1, \text{ when the feature is selected} \\ 0, \text{ when the feature is deselected} \end{cases} \tag{4}$$

$a \subseteq A$ where $|A| = 2^m - 1$ and.$a = (a_1 a_2 \cdots .a_t)$

4.4.4. Agent

The modelling of feature selection will be based on multi-agent reinforcement learning. Each reinforcement learning agent i^{th} is responsible on making a decision of selecting a subset of features based on the state. The agent is supposed to capture the knowledge generated from the subsequent instances of records, states $s_{i,\,t}$, action $a_{i,\,t}$, next state $s_{i,\,t+1}$, and the reward $r_{i,\,t}$ obtained by the taken action. To enable the learning of the agent, we use a replay buffer for storing the instances ($s_{i,t}$, $a_{i,t}$, $s_{i,t+1}$, $r_{i,t}$) that resulted after agents take different actions several times. After that, and according to our batch learning method in Section 5.4.6, we sample the stored instances from the replay buffer for training each agent's Deep-Q-network (DQN).

4.4.5. Reward scheme

The role of reward distribution is to assign to each agent a reward value equivalent to its contribution to changing the overall prediction accuracy. Considering that the agents are operating simultaneously, it is difficult to separate the contribution of each agent. Consequently, a non-stationary environment will occur. To resolve this, we propose different methods for reward distribution. Each of these methods is based on changing the decision within certain number of iterations and calculating the corresponding accuracy that is resulted from the overall decisions of all agents. We assume that the set of combinations of the various decisions is $a = (a_{i,1} a_{i,2} \cdots a_{i,3} ... a_{i,k})^T$ where each a denotes the joint action of all agents. In addition, we assume that the corresponding accuracy of each decision is given as $Ac = (ac_1 ac_2 \cdots ac_i ... ac_k)^T$ where k denotes the number of combinations. The reward for each agent is derived from one of the four suggested mathematical models. The collection of all combinations with the corresponding accuracies is added to a matrix named as a combinatory matrix (CM) that is shown in Eq. (5). Fig. 3 provides a conceptual diagram of our proposed framework for fair reward distribution based on *CM*.

$$CM = \begin{pmatrix} a_{1,1} & a_{2,1} & . & a_{m,1} & ac_1 \\ a_{1,2} & a_{2,2} & . & a_{m,2} & ac_2 \\ & . & . & . & . & . \\ & . & . & . & . & . \\ a_{1,k} & a_{2,k} & . & a_{m,k} & ac_k \end{pmatrix} \tag{5}$$

4.4.5.1. Accumulative contribution (ACC). This method is proposed to calculate the reward of each agent based on their importance contribution to the actions taken. In this method, the reward for each agent is calculated as an aggregation of the various decisions multiplied by the aggregation of the accuracies. This is applied when the decision is changing. However, when the decision is fixed and the average accuracy is below certain threshold (*Thr*), then the reward is discounted with the factor *w* that denotes the window length, as shown in Eq. (6). This is achieved by utilizing a sliding window with a specific length size *w*, the sliding window crosses over the CM to calculate the reward contribution for each agent.

$$r_j = \begin{cases} \frac{\sum_{i=1}^{k} a_{i,j} \times \sum_{i=1}^{k} ac_j}{w} \; if \sum_{i=1}^{k} a_{i,j} == 0 \; or \sum_{i=1}^{k} a_{i,j} == w \; and \frac{\sum_{i=1}^{k} ac_j}{w} < Thr \\ \sum_{i=1}^{k} a_{i,j} \times \sum_{i=1}^{k} ac_j \; otherwise \end{cases} \tag{6}$$

4.4.5.2. Alternative contribution (ALC). The second method proposed in this study for reward distribution is Alternative Contribution (ALC). This method aims to distribute the reward values according to the frequency of decision changes and the relation of this frequency with the accuracy changes. The reward calculation according to this method is done using Eq. (7) and (8). The goal of Eq. (7) is to measure the effect of accuracy change concerning decision change, and Eq. (8) is to perform probabilistic scaling of the rewards.

$$r_j = \frac{\sum_{i=1}^{K} (a_{i,j} - a_{i,j-1})(ac_j - ac_{j-1})}{\sum_{i=1}^{K} (a_{i,j} - a_{i,j-1})} = \frac{\sum_{i=1}^{K} da_{i,j} dac_j}{\sum_{i=1}^{K} da_{i,j}} \tag{7}$$

$$sm_j = sm(r_j) = \frac{e^{r_k}}{\sum_{k=1}^{M} e^{r_k}} \tag{8}$$

Where:
$da_{i,j}$ denotes the differentiation of action over time.
dac_j denotes the differentiation of accuracy over time.
sm represents softmax normalization.

For more elaboration, we present the calculation of the reward of the agent in the two methods, ACC and ALC, in Fig. 4. As it is shown, in the case of ACC, the reward is calculated as the multiplication of two areas: the area under the accuracy curve (*Ac*) and the area under the action curve. The only exception is that when the decision is fixed, the area under the action curve is 0 or 1. In this case, we divide the calculated reward by *w*, which represents the length of the window. On the other side, in the case of alternative contribution ALC, the reward is represented geometrically by summation of samples from the curve of accuracy derivative dac_j multiplied by the derivative of actions da_i and divided over the derivative of actions. Next, the rewards are scaled by applying the SoftMax function *sm*, as shown in Eq. (8).

4.4.5.3. Impurity (IM). The third method of the proposed FARD-DFS uses a regression model based on the selected features and their corresponding labels. This information is considered after training

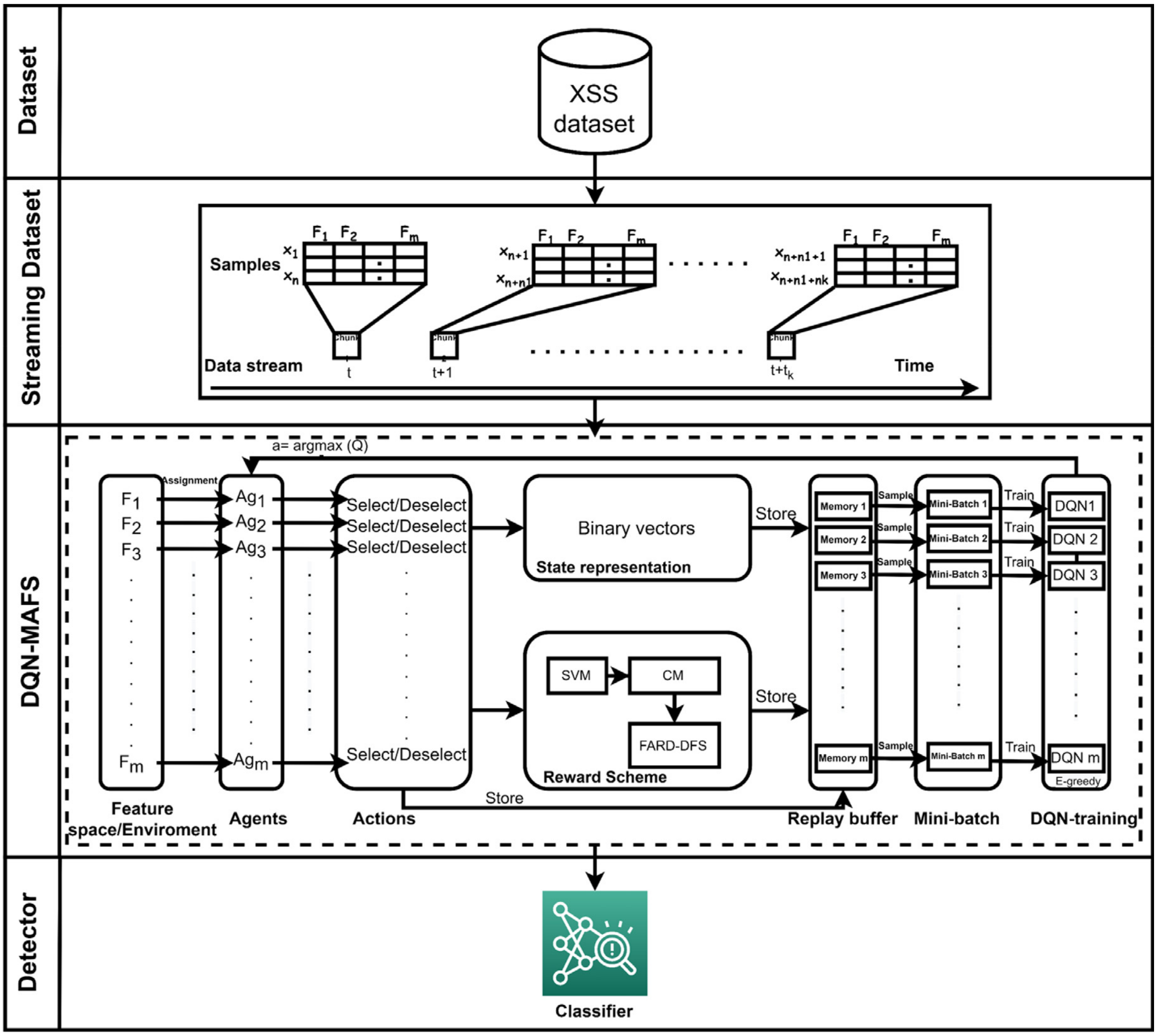


Fig. 2. The overall methodology.

as the importance of feature and is used as a reward given the information of other features as well as the information of labels. This article uses random forest (RF) with feature impurity to calculate the reward. We chose the RF because it has a robust built-in feature-importance measurement. In this case, variance reduction (V) is used as a measure of impurity, as shown in Eq. (9). The lowest impurity is equivalent to a higher reward. Also, this method is based on CM to calculate the impurity of the selected features by agents' action.

$$V = \frac{1}{k}\sum_{i=1}^{k}(y_i - \mu)^2 \tag{9}$$

Where:

y_i is label for an instance, k denotes the number of instances and μ is a value obtained by the mean $\frac{1}{k}\sum_{i=1}^{k} y_i$.

4.4.5.4. One action at one time (OA-OT). The last proposed method in this work for reward distribution is named as One-Action at One-Time (OA-OT). This method avoids the challenge of reward distribution by allowing one agent to change a decision at one time. In other words, at one time unit t, it is only permitted one agent to make a decision change. Mathematically, this is described as:

$$\forall t, \sum_{i=1}^{N}|a_i(t) - a_i(t-1)| = 1 \tag{10}$$

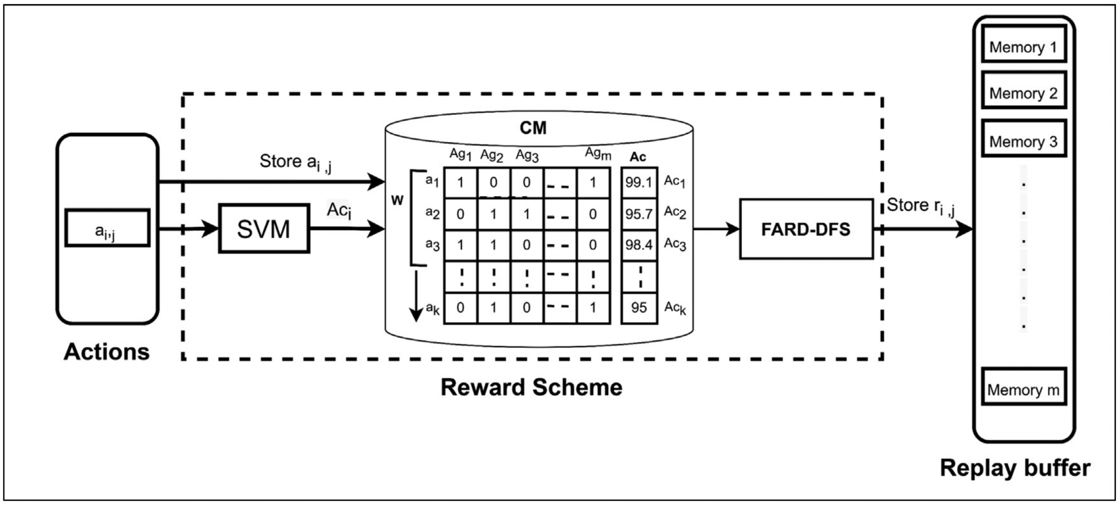
Where:

N denotes the number of agents, $a_i(t)$ denotes the decision of agent i at moment t

Hence, the change amount of the accuracy after the action is used as a reward value for the subject agent. To accomplish this, we add a restriction to the combinatory matrix (CM) Eq. (5) to only have hamming distance between two consecutive rows equal to 1; according to that, a new CM will be generated based on this method and the action policy.

4.4.6. Batch learning for training DQN

The buffer inside the agent contains a wide range of records that are combined of state, action, next state, and reward. They cannot be used all for training. Hence, a subset of them is used for this purpose. In order to enable effective training, we build a statistical distribution for sampling from the buffer to train DQN. The statistical distribution will give a higher probability for selecting samples subject to misclassification in previous trials. Assuming that the



	Ag_1	Ag_2	Ag_3		Ag_m	Ac	
a_1	1	0	0	- -	1	99.1	Ac_1
a_2	0	1	1	- -	0	95.7	Ac_2
a_3	1	1	0	- -	0	98.4	Ac_3
	⋮	⋮	⋮	- -	⋮	⋮	
a_k	0	1	0	- -	1	95	Ac_k

Fig. 3. Our proposed sub-model for DQN-MAFS, which is used for fair reward distribution to agents and is designated as fair agent reward distribution based dynamic feature selection FARD-DFS.

buffer size is N, then the initial probability selection of each sample is $\frac{1}{N}$.

When the sample is selected, it is fed again for evaluation, and a counter $C(index)$ is associated with each sample $index$:

$$C(index) = \begin{cases} C(index) + 1, if\ the\ reward\ has\ increased \\ C(index), else \end{cases} \quad (11)$$

Next, the probability of selection is calculated as

$$P(index) = \frac{C(index)}{\sum_{i=1}^{M} C(i)} \quad (12)$$

The increase of the reward is measured by the difference between the current reward $r_{i,t}$ and the reward after training $r_{i,t+1}$ based on the batch and making a decision.

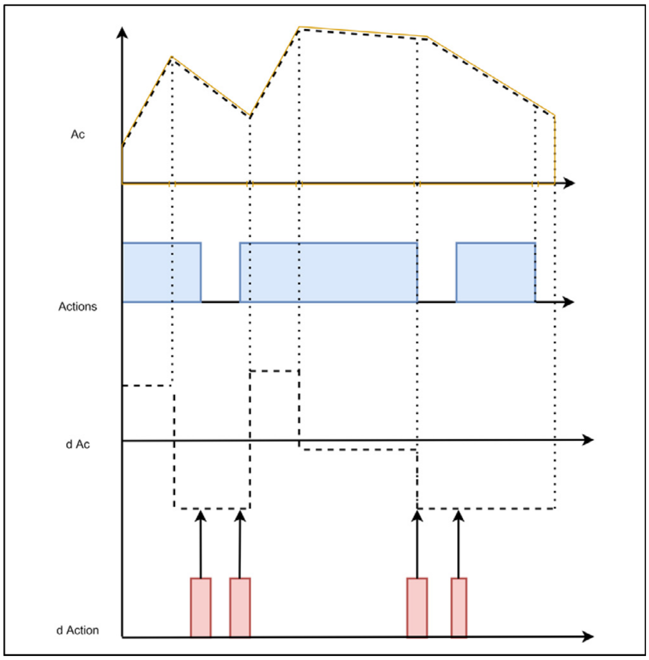


Fig. 4. Reward calculation for ACC and ALC approaches.

4.4.7. General DQN-MAFS algorithm

The agents take their own action decisions based on their policy networks $\pi_i : s \to p(a_i)$. The policy π_i denotes the transition probability that identifies which action a_i should be taken by an agent to evolve the feature sub-space from state s_t to s_{t+1}, and each agent i should receive a reward $r_{i,t}$. For exploring more possibilities for selecting features, we utilise the $\varepsilon - greedy$ policy in this study, as shown in Eq. (13). Algorithm 1 shows the pseudocode of the $\varepsilon - greedy$ policy.

$$\pi(s_{i,t}, a_{i,t}) = \begin{cases} random\ a_{i,t}, \varepsilon \\ \arg max_a Q^{\pi}(s_{i,t}, a_{i,t}; \theta), 1 - \varepsilon \end{cases} \quad (13)$$

Random $a_{i,t}$ denotes the agent has transition probability ε to make random action at state $s_{i,t}$, whilst in case the agent has the transition probability $1 - \varepsilon$ to take action $a_{i,t}$ by exploiting the maximum action value according to the output of the evaluation network that provides the maximum expected reward. This step enables the proposed approach to balance the exploration and exploitation processes (Z. Fang et al., 2019).

Algorithm 1 (*Epsilon greedy action*).

```
Input: (1) expectedReward: list of expected rewards for each
    possible action
    (2) epsilon: float number between 0 and 1, refers to
    exploration probability.
    Output: (1) action: an integer represents the index of the
    best action.
    Start Algorithm
    action = None
    if random() <= epsilon then
    action = randInteger()
    else
    action = argmax(expectedReward)
    end if
    End Algorithm
```

Thus, in the training DQN process, each agent trains its policy networks via sampling its instances from their replay buffer separately using our proposed model given in Sub-section 4.4.2. Then feed them to the DQN for training in order to achieve the maximum action value and maximum long-term expected reward value according to Bellman Equation (Barto, 2018).

$$\mathcal{Q}(s_{i,t}, a_{i,t}; \theta_t) = r_{i,t} + \gamma \max \mathcal{Q}(s_{i,t+1}, a_{i,t+1}; \theta_{t+1}) \quad (14)$$

Where θ_t is the parameters of the Q network setting (evaluation network), max $\mathcal{Q}(s_{i,\,t+1},\ a_{i,\,t+1})$ denotes the maximum action value according to the $\varepsilon - greedy$ policy, and γ represents the discount factor. The sampled transition from the replay buffer $(s_{i,\,t},\ a_{i,\,t},\ s_{i,\,t+1},\ r_{i,\,t})$ is input into the Q network, in which $s_{i,\,t}$ is fed into the evaluation network to obtain the Q value $\mathcal{Q}(s_{i,\,t},\ a_{i,\,t};\ \theta)$ of the action $a_{i,\,t}$, while the $s_{i,\,t+1}$ is inputted to the target network to achieve the maximum Q value $max\mathcal{Q}(s_{i,\,t+1},\ a_{i,\,t+1};\ \theta_{t+1})$. Then the expectation $y_{i,\,t}$ of the reward is obtained based on the output of the target network via calculating the maximum Q value as demonstrated in Eq. (15).

$$y_{i,t} = \begin{cases} r_{i,t},\ \textit{if step ends at } t+1 \\ r_{i,t} + \gamma maxQ(s_{i,t+1}, a_{i,t+1}; \theta_{t+1}),\ \textit{otherwise} \end{cases} \quad (15)$$

In order to set the parameters of the Q network, the gradient descent method is used. Thus, the parameters of the Q network can be updated frequently to calculate the Q value accurately. The loss function $L(\theta)$ is calculated according to expectation value $y_{i,\,t}$, and Q value $\mathcal{Q}(s_{i,\,t},\ a_{i,\,t};\ \theta)$ shown in Eq. (16).

$$L(\theta) = (y_{i,t} - \mathcal{Q}(s_{i,t}, a_{i,t}; \theta))^2 \quad (16)$$

The evaluation network's parameters are modified by calculating the derivative of the θ as shown in Eq. (17).

$$\theta \leftarrow \theta + \alpha(y_{i,t} - \mathcal{Q}(s_{i,t}, \acute{a}_{i,t}; \theta)) \nabla\, \mathcal{Q}(s_{i,t}, a_{i,t}; \theta) \quad (17)$$

To preserve the stability of the proposed method that may be impacted by the rapid updating of Q network's parameters, the target network's parameters θ_{t+1} are derivative from the parameters of the evaluation network θ at every end of the epoch in training. Hence, after the given number of epochs, the process of parameter updating will be finished.

Algorithm 2 (*DQN-MAFS algorithm*).

```
Input: Chunks of Dataset, list of feature spaces
  $f = \{f_1, f_2, f_3, \cdots, f_m\}$, window w
  Output: selectedFeatures
  Start Algorithm:
  Initialize replay buffer size (RB)
  Initialize multi-agents RL equal to number of features
  Initialize $\mathcal{Q}$ action_value with weights parameter $\theta$ for
  evaluation network
  Initialize $\acute{\mathcal{Q}}$ action_value with weights parameter
  $\theta_{t+1} \leftarrow \theta$ for target network
  for everyChunk
    for t = 1 to T do
      for each agent
        Using a classifier to produce CM based on Eq. (5) for
        selected features based on window w
        for n = 1 to N sample in CM do
          Using FARD-DFS approaches to observe the reward $r_t$
          for each agent i
          Assign generated $r_{i,t}$ for each agent based on their
          actions $a_{i,\,t}$ and store in RB
        end for
        Sample batches of transitions $(s_{i,\,t},\ a_{i,\,t},\ s_{i,\,t+1},\ r_t)$
        Calculate the target value of the target network using
        Eq. (15)
        Perform gradient descent using Eq. (16)
        action $a_t = \mathrm{argmax}_a\ \acute{\mathcal{Q}}(s_{i,\,t},\ a_{i,\,t};\ \theta)$ based on Eq. (13)
      end for
    end for
    Add action to selectedFeatures
    Update $\varepsilon$, reset $\theta_{t+1} \leftarrow \theta$
  end for
  End Algorithm
```

4.5. Detector

The last component of our proposed framework is the classification process implemented in this study since it is crucial for predicting XSS attacks. Thus, a set of classifiers are adopted to classify the records based on the selected optimal features provided sequentially from the previous stage DQN-MAFS for each chunk. Then the current knowledge of the classifier is updated with new knowledge for the next chunk at the moment $t+1$ or whenever a new data stream is available. In our framework, we evaluate DFS on four classifiers, namely, support vector machine (SVM), K-nearest neighbour (KNN), decision tree (DT), and logistic regression (LR).

5. Experimental works and results

This section presents the experimental design, dataset, time series visualization, comparison with other benchmarks, and overall discussion.

5.1. Experimental design

The experiments were performed using four real XSS datasets, namely, D1-66, D2-167, D3-30, and D4-30. The two benchmarks used for comparison are Tariq et al., (2021) and K. Liu et al., (2021). We benchmarked our work with Tariq et al., (2021) because of their adaptive XSS detection method based on genetic algorithm and RL, while the method of K. Liu et al., (2021) because of their dynamic feature selection based on multi-agent RL approach. Both these two benchmarks are re-implemented in a similar design to our method to reproduce their results using four datasets, as mentioned earlier. For evaluation, each dataset was partitioned into chunks, similar to how data streams feed incremental learning models. The method was required to provide a different feature selection decision for each chunk. The total number of chunks is 9 for each dataset. The dataset splitting for training and testing depends on how many data samples we have. In this article, all used datasets are divided into 0.8 for training and 0.2 for testing. This is due to the fact that we have divided each of the used datasets into several chunks in which the number of the data in each chunk is small; thus, we select the highest percentage of the data samples for training to generalize the model and avoid the under-fitting of the model. The chunks are provided to each of the agents simultaneously at the specific moment as well as to benchmarks to guarantee objective evaluation. The comparisons are evaluated in terms of mean and maximum accuracy, precision, recall, and F1-measure. All these metrics are based on confusion matrix parameters which are True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Hence, the derivation of the performance metrics mentioned above metrics are listed in Eq. (18–21).

$$Accuracy = \frac{TP + TN}{(TP + TN + FP + FN)} \quad (18)$$

$$Precision = \frac{TP}{(TP + FP)} \tag{19}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{20}$$

$$F1 - measure = 2 \times \frac{(Recall \times Precision)}{(Recall + Precison)} = \frac{2TP}{2TP + FP + FN)} \tag{21}$$

All the experiments were carried out using Microsoft Azure for code execution with GPU Tesla 24 cores and 256 GB RAM. In our experiments, we set the DQN-MAFS with several parameters. Each agent has its own deep Q-network; the batch learning is set as shown in Eq. (11–12). We use the $\varepsilon-$ greedy policy to balance between the exploration and exploitation processes. The value $\varepsilon - greedy$ policy is 0.97 and the ε decay value is 0.01 from 1 to 0 at the end of each episode. For the experience replay, the capacity of the replay buffer is set to 800. The Q-network is set in this approach as three neural network layers. ReLU is used as the activation function at the hidden layers 2nd and 3rd layers with 32 and 16 neurons, respectively. Whilst the loss function is mse, and the optimizer is Adam. The total number of epochs is 5, with a batch size of 32. The learning rate α is 0.001, and the discount factor γ is 0.995. The four predictors (SVM, KNN, DT, and LR) are set with different parameters. SVM is set with radial basis function (RBF) kernel, maximum iteration = 8000, regularization proportion c = 0.2, and probability = true. Also, five neighbours are used with the KNN. Furthermore, LR is set with a maximum iteration equal to 500. Finally, the DT is used with 15 estimators, and the random generator is initialized with 42. In the ACC and ALC methods, we used a sliding window with a size equal to 4 steps to calculate the reward contribution from the CM. The ALC method's threshold value is set at a 0.6 of accuracy rate.

5.2. Dataset

According to (Melicher & Fung, 2021; Nunan et al., 2012), there is lacking of real-world benchmarking XSS datasets. Nevertheless, we came across four XSS attacks datasets used in this study with a purpose of demonstrating the efficiency of our proposed approach when it comes to the low and high number of dimensions; we summarize them in Table 3, and we give details of each of them in the following sub-sections.

5.2.1. Datasets of (Mokbal et al., 2019) (D1-66) and (Mokbal et al., 2021) (D2-167)

The first dataset, D1-66, was published by (Mokbal et al., 2019) and updated in (Mokbal et al., 2021) and used in this study as the second dataset, D2-167. These two datasets are considered the first comprehensive cross-site scripting-based numerical features dataset extracted from real XSS attacks. The benign samples in this dataset were created by crawling the top 50,000 websites ranked by Alexa. In contrast, the malicious samples were collected by crawling raw XSS repositories such as XSSed and Open Bug Bounty.[8] Initially, the author extracted only 66 numerical features. Then (Mokbal et al., 2021) extracted more comprehensive features, were extracted 167 numerical features. Both datasets have been available online recently on GitHub.[9] Furthermore, these datasets consist of three types of features: HTML, JavaScript, and URL.

5.2.2. Dataset of (Zhou & Wang, 2019) (D3-30)

The third dataset D3-30 was created by (Zhou & Wang, 2019). This dataset was gathered from multiple GitHub sources and security forms and contained around 6503 normal records and 3497 XSS payloads. Then 30 script and URL context features were obtained using open-source ontology modelling tools (Protégé) based on only four characteristics.

5.2.3. Dataset of (Y. Fang et al., 2018) (D4-30)

The fourth dataset used in this study is D4-30 which was initially collected by (Y. Fang et al., 2018) as a raw dataset from different sources, namely, GitHub[10] and XSSed. Then the features were extracted directly from the XSS payload by (Tariq et al., 2021) based on (Zhou & Wang, 2019); here also, only 30 features were extracted using the JSoup parser library. The raw dataset is collected from GitHub[11] and XSSed. Both datasets D3-30 and D4-30 are available on.[12]

5.3. Time series visualization

The first set of results provides time series of both accuracy and F1-measure. The latter considers both recall and precision by measuring their harmonic mean. In order to evaluate our proposed method of dynamic feature selection for evolved XSS detection, four classifiers were used, namely, SVM, KNN, DT, and LR. Thus, we select the largest dataset in terms number of features, i.e., D2-167, and provide its time series behaviour in Fig. 5. It is observed that each method has a variable accuracy and F1-measure according to the chunk. However, in all methods, the minimum and maximum values were in the range of 85.79 % – 98.81 % and 71.45 % – 97.84 % for accuracy and F1-measure, respectively. Another observation is that a drop in the accuracy and F1 measure to the values 85.79 % and 71.45 %, 86.51 % and 73.09 %, respectively, has occurred in chunk 7 with IM and ALC methods. On the other side, another drop has shown in the accuracy and F1 measure to 92.14 % – 84.45 % respectively, within the ACC method in chunk 3. Furthermore, OA-OT approach has less steady performance compared with other methods. The maximum achieved accuracy, recall, F1-measure, and AUC were 98.81 %, 97.70 %, 97.84 %, and 98.41 %, respectively; they were obtained using a DT classifier. Overall, all methods have provided an outstanding performance with four classifiers used despite the variation. This is interpreted by the fact that each rewarding method focuses on one aspect that can be captured in a specific chunk and lost partially in others.

5.4. Comparison with other studies

We compare our proposed method with two studies Tariq et al., (2021) that have proposed an adaptive method to detect the new XSS attacks; this method was tested using two datasets, D3-30 and D4-30 datasets. The second work is K. Liu et al., (2021), which proposed a method for dynamic feature selection based on MARL. This comparison is conducted using four different datasets with various feature sizes in each dataset. The goal of using different sizes and numbers of features for the data is to explore the performance variations according to these factors. According to the previous results., we selected the DT classifier as a predictor with our proposed approach because of its results' superiority over other used classifiers such as SVM, KNN, and LR. In the subsequent sections, we present the analysis of the results for the mentioned datasets.

5.4.1. Results of D1-66 dataset

Another evaluation was performed on 66 features dataset is provided in Table 5. As it is shown, we find that the best achieved

[8] https://www.openbugbounty.org/.
[9] https://github.com/fawaz2015/XSS-dataset.
[10] https://github.com/duoergun0729/1book/tree/master/data.
[11] https://github.com/duoergun0729/1book/tree/master/data.
[12] https://github.com/IramTariq/XSS-attack-detection/tree/master/Testing_Data.

Table 3
Existing XSS benchmarking datasets.

Dataset name	Author name	No. of samples	No. of features	Source of malicious samples	Source of benign samples	Collection method
D1-66	(Mokbal et al., 2019)	100,000 benign samples, and 38,569 malicious samples	66 features	XSSed and Open Bug Bounty	50,000 Top Alexa websites	Crawling and parsers
D2-167	(Mokbal et al., 2021)		167 features			
D3-30	(Zhou & Wang, 2019)	6503 normal records and 3497 XSS payloads	30 features	XSSPL and XSSed	GitHub	Ontology modeling tools (Protégé)
D4-30	(Y. Fang et al., 2018)	135,507 normal records and 16,151 XSS payloads.	30 features	XSSed	GitHub	JSoup parser library

mean and maximum accuracy was for FARD-DFS-IM with values of 99.59 % and 100 %, respectively. Furthermore, we find that the mean and maximum achieved F1-score were for FARD-DFS OA-OT and FARD-DFS-IM, with values of 90.75 % and 100 %, respectively. Furthermore, we observe that the accuracy of all FARD-DFS variants is always higher than 99 %. Hence, we readily conclude that FARD-DFS is superior. In addition, we observe that FARD-DFS-IM has provided the least standard deviation (Std) in accuracy 0.44 %, which is an indicator of more expected performance.

5.4.2. Results of D2-167 dataset

The second used dataset has a high number of features, which is 167. Hence, the evaluation based on this dataset tests the model's capability to perform on a large number of extracted features. Table 6 shows that the most superior method in terms of accuracy and F1-score is FARD-DFS OA-OT. The reached mean and maximum accuracy rates were 97.66 % and 98.81 %, respectively and the reached mean and maximum F1-score were 95.81 % and 97.84 %, respectively. This shows an overall outperformance of FARD-DFS OA-OT over the remaining models.

Furthermore, it is observed that all variants of FARD have provided good performance compared with other models. For example, the maximum accuracy was always higher than 98 % for all FARD variants. This provides the capability of FARD to attain needed dynamic knowledge for selecting the most relevant features compared with other models. Furthermore, The least accuracy and F-measure have been observed at Tariq et al., (2021) with a value of 87.19 % and 88.93 %.

In order to capture the consistency in the performance, we calculated the standard deviation. The lower values of the latter indicate to more expected performance. It is observed that the least std has occurred in OA-AT for accuracy, which was 0.77 % compared with the std of Tariq et al., (2021) 13.21 %.

5.4.3. Results of D3-30 dataset

The analysis of obtained results of our model and the benchmarks based on the D3-30 dataset is detailed in Table 7. It can be found that the best-achieved accuracy, a mean of 98.28 %, and a maximum value of 100 % were for FARD-DFS-IM and K. Liu et al., (2021), respectively. Furthermore, the best-achieved F1-measure in terms of mean was for FARD-DFS-ACC and in terms of maximum value was for K. Liu et al., (2021) for values of 97.38 % and 100 %, respectively. This implies that FARD-DFS has better performance overall than the benchmarks regarding accuracy and F-measure. Another observation is that Tariq et al., (2021) have a recall for both mean and maximum values of 99.97 % and 100 %, respectively. However, FARD-FDS is better at balancing true positive and negatively predicted results, which means more reduction of false alarms and attaining good optimistic predictions. Lastly, we observe that FARD-DFS-ACC has achieved the least std, which is 0.51 %. Again, this is an indicator of a better estimation of the performance of the approach.

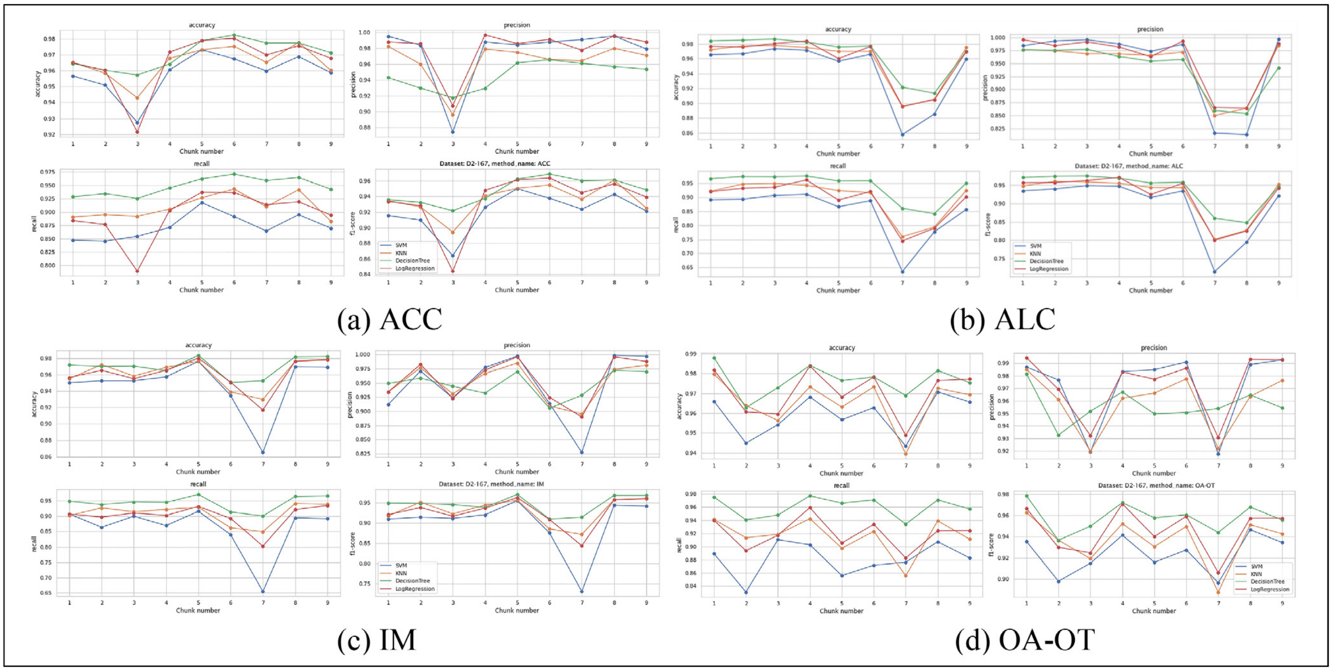
5.4.4. Results of D4-30 dataset

The results of comparing FARD-DFS with other methods for the D4-30 dataset are presented in Table 8. It is found that FARD-DFS-IM and FARD-DFS-ACC obtain the best mean and maximum accuracy with values of 98.37 % and 99.01 %, respectively. Similarly, the best mean and maximum achieved F1-score are 98.42 % and 99.10 % for FARD-DFS-IM and FARD-DFS-ACC, respectively. K. Liu et al., (2021) has provided a higher precision; however, the recall was relatively low 95.63 % for mean. On the hand, Tariq et al., (2021) has accomplished a higher mean recall of 99.99 %; however, the precision was relatively low, 93.32 %. This shows that FARD-DFS was generally superior to both K. Liu et al., (2021) and Tariq et al., (2021) from two perspectives of positive and negative predictions. Lastly, the minimum std for accuracy is observed at FARD-DFS-ACC, which is 0.48 %. This is an indicator of more expected performance. Another observation that can be made from this dataset results and the previous ones is that the accuracy was higher than recall in general. This is a common observation when the dataset contains more negative samples than positive ones. This leads to higher accuracy than recall because the model tends to predict negative samples which is not captured by the recall.

5.4.5. Features and memory reduction

The percentage of memory reduction measures the effectiveness of feature selection. In order to measure memory reduction, we take the percentage of deselected features as an indicator of memory reduction. This is given for two datasets, namely, the one with the highest number of features, i.e.,167, which is D2-167, and the second one with the minimum number of features, i.e., 30, which is D3-30. For the memory reduction in the D2-167 dataset, the results are presented in Table 9. The results show that the highest reduction has occurred for FARD-DFS-ALC, ranging from 65 % to 49 % within all chunks. This indicates that our methods are most effective than the method that used in the work of Mokbal et al., (2021) with same dataset that reduced the memory statically by 60 %, thanks to using hybrid approach that consists of two most popular feature selection approaches are Information Gain (IG) integrated with the Sequential Backward Selection method (SBS). For the memory reduction in D3-30, the results are introduced in Table 10. As can be seen, the maximum memory reduction is obtained by FARD-DFS-ACC, with the percentage ranging from 67 % to 33 % in all chunks. Whilst both Tariq et al., (2021) and Zhou & Wang, (2019) used the same dataset and selected entire features based on the knowledge and without any memory reduction attempts.

For more elaboration, we present a comparison between the methods in terms of the number of selected features for D2-167 and D3-30 datasets in Fig. 6. It is confirmed that ALC has accomplished the least number of selected features for every chunk with respect to D2-167. Conversely, ACC has accomplished the least number of selected features in almost chunks of data stream with respect to D3-30, except in 1st, 3rd, and 7th chunks, where ALC has



(a) ACC (b) ALC

(c) IM (d) OA-OT

Fig. 5. Time series virtualization results of our developed FARD-DFS in terms of accuracy, precision, recall, and f1-measure metrics, by using D2-167 datasets.

Table 5
Numerical comparison of our developed FARD-DFS and its comparison with Tariq et al., (2021) and K. Liu et al., (2021) by using D1-66 dataset.

Methods Name	Accuracy			Precision			Recall			F1-measure		
	Mean	Max	Std	Mean	Max	Std	Mean	Max	Std	Mean	Max	Std
FARD-DFS-ACC	99.42 %	99.82 %	0.57 %	86.04 %	96.77 %	16.89 %	85.21 %	97.22 %	22.45 %	85.27 %	97.00 %	20.23 %
FARD-DFS-ALC	99.06 %	99.82 %	1.13 %	85.03 %	95.00 %	12.09 %	81.36 %	95.00 %	16.70 %	82.98 %	94.74 %	14.32 %
FARD-DFS-IM	**99.59 %**	**100 %**	**0.44 %**	89.91 %	**100 %**	15.40 %	88.50 %	100 %	9.62 %	88.62	**100 %**	11.31 %
FARD-DFS OA-OT	99.52 %	99.91 %	0.45 %	89.79 %	100 %	7.59 %	**91.90 %**	**100 %**	7.30 %	**90.75 %**	97.44 %	6.79 %
K. Liu et al., (2021)	94.88 %	98.30 %	2.75 %	**96.58 %**	99.85 %	**3.91 %**	84.69 %	93.58 %	**6.67 %**	90.20 %	96.61 %	**5.44 %**
Tariq et al., (2021)	87.49 %	94.80 %	9.57 %	84.58 %	94.40 %	19.02 %	90.44 %	97.86 %	7.33 %	85.94 %	94.63 %	14.09 %

Table 6
Numerical comparison of our developed FARD-DFS and its comparison with Tariq et al., (2021) and K. Liu et al., (2021) by using D2-167 dataset.

Methods Name	Accuracy			Precision			Recall			F1-measure		
	Mean	Max	Std	Mean	Max	Std	Mean	Max	Std	Mean	Max	Std
FARD-DFS-ACC	97.06 %	98.27 %	0.92 %	94.68 %	96.58 %	1.73 %	**94.80 %**	97.09 %	1.67 %	94.78 %	96.83 %	1.63 %
FARD-DFS-ALC	96.64 %	98.67 %	2.81 %	94.00 %	97.72 %	4.86 %	93.99 %	97.57 %	5.12 %	93.99 %	97.53 %	4.98 %
FARD-DFS-IM	96.99 %	98.38 %	1.22 %	94.77 %	97.19 %	2.26 %	94.39 %	97.12 %	2.37 %	94.58 %	97.05 %	2.23 %
FARD-DFS OA-OT	**97.66 %**	**98.81 %**	**0.77 %**	95.63 %	98.16 %	1.37 %	96.00 %	97.70 %	**1.57 %**	**95.81 %**	**97.84 %**	**1.34 %**
K. Liu et al., (2021)	95.56 %	97.01 %	0.84 %	**99.36 %**	**99.85 %**	**0.39 %**	84.67 %	89.57 %	2.83 %	91.41 %	94.30 %	1.67 %
Tariq et al., (2021)	87.19 %	95.38 %	13.21 %	91.99 %	95.93 %	1.86 %	87.80 %	**99.54 %**	16.55 %	88.93 %	95.20 %	9.38 %

selected the least number of features in the earlier chunks. This confirms the results of memory reduction provided earlier.

5.5. Overall discussion

We present a comparative analysis of the mean and max accuracy for all methods in Table 11. The results provide that majority of values of FARD-DFS are superior over the benchmarks K. Liu et al., (2021) and Tariq et al., (2021). Another observation is that FARD-DFS OA-OT is more superior when the number of features is high compared with FARD-DFS-ACC, which has behaved better for a low number of features around 30, while FARD-DFS-IM has behaved better with a medium number of features around 66. In addition, we find that both K. Liu et al., (2021) and Tariq et al., (2021) have a degradation in the performance when the number of features increases, which means less scalability. This can be interpreted by the powerfulness of the reward calculation and distribution in FARD-DFS. Furthermore, the dynamicity handling of FARD-DFS is consistent and promising when it is used against the evolving nature of XSS attacks in an incremental or stream learning manner.

As an assessment step for demonstrating the statistical significance of our proposed FARD-DFS approaches' results compared to benchmark methods, we conducted some statistical significance tests. Firstly, we applied a *t*-test mechanism on the accuracy and F1-measure results of the D2-167 dataset for each method in all nine chunks compared with K. Liu et al., (2021) and Tariq et al., (2021), as shown in Table 12. Observing the table, we find that a statistical significance with a confidence level of more than 90 % has been achieved to support the superiority of our proposed

Table 7
Numerical comparison of our developed FARD-DFS and its comparison with Tariq et al., (2021) and K. Liu et al., (2021) by using D3-30 dataset.

Methods Name	**Accuracy**			**Precision**			**Recall**			**F1-measure**		
	Mean	**Max**	**Std**	**Mean**	**Max**	**Std**	**Mean**	**Max**	**Std**	**Mean**	**Max**	**Std**
FARD-DFS-ACC	**98.28 %**	99.00 %	**0.51 %**	98.53 %	100 %	1.64 %	96.29 %	98.65 %	1.82 %	**97.38 %**	98.67 %	**0.93 %**
FARD-DFS-ALC	98.00 %	99.50 %	1.12 %	98.95 %	100 %	2.08 %	95.13 %	98.59 %	2.32 %	96.98 %	99.29 %	1.69 %
FARD-DFS-IM	97.50 %	99.00 %	1.48 %	98.36 %	100 %	1.89 %	94.32 %	97.33 %	2.66 %	96.27 %	98.65 %	2.03 %
FARD-DFS OA-OT	97.83 %	99.50 %	3.72 %	**99.77 %**	**100 %**	**0.68 %**	93.90 %	98.64 %	9.97 %	96.51 %	99.32 %	6.24 %
K. Liu et al., (2021)	97.00 %	**100 %**	1.41 %	98.32 %	100 %	3.33 %	92.59 %	**100 %**	3.14 %	95.32 %	**100 %**	2.32 %
Tariq et al., (2021)	93.48 %	98.57 %	5.77 %	86.98 %	97.25 %	11.55 %	**99.97 %**	100 %	**0.05 %**	92.63 %	98.55 %	7.16 %

Table 8
Numerical comparison of our developed FARD-DFS approaches and its comparison with Tariq et al., (2021) and K. Liu et al., (2021) by using D4-30 dataset.

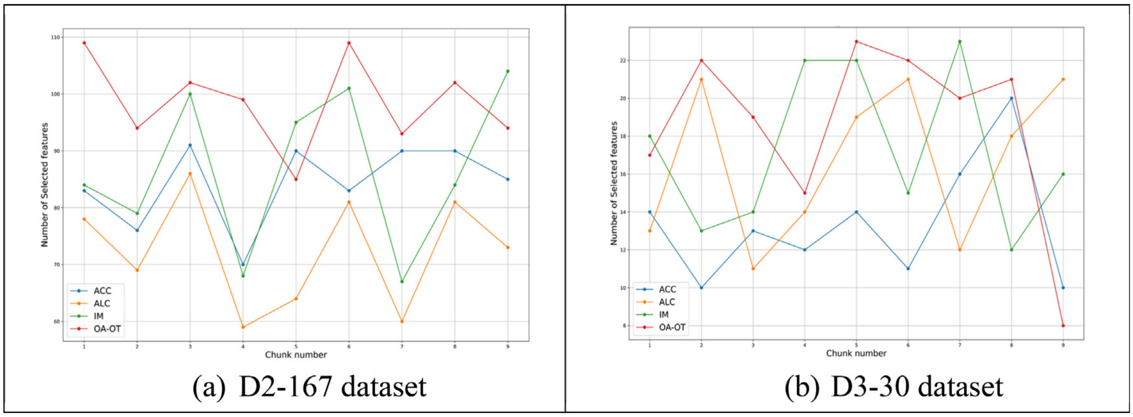
Methods Name	**Accuracy**			**Precision**			**Recall**			**F1-measure**		
	Mean	**Max**	**Std**	**Mean**	**Max**	**Std**	**Mean**	**Max**	**Std**	**Mean**	**Max**	**Std**
FARD-DFS-ACC	98.29 %	**99.01 %**	**0.48 %**	**99.52 %**	**100 %**	**0.47 %**	97.20 %	98.36 %	0.83 %	98.35 %	**99.10 %**	**0.47 %**
FARD-DFS-ALC	96.39 %	98.74 %	2.97 %	99.13 %	**100 %**	0.91 %	93.95 %	97.86 %	5.71 %	96.38 %	98.85 %	3.18 %
FARD-DFS-IM	**98.37 %**	98.98 %	0.49 %	99.57 %	**100 %**	0.60 %	97.31 %	98.29 %	0.63 %	**98.42 %**	99.06 %	0.47 %
FARD-DFS OA-OT	97.98 %	99.06 %	0.84 %	99.43 %	**100 %**	0.69 %	96.70 %	98.39 %	1.74 %	98.04 %	99.09 %	0.86 %
K. Liu et al., (2021)	97.36 %	98.27 %	0.58 %	99.32 %	**100 %**	0.73 %	95.63 %	98.00 %	1.49 %	97.43 %	98.42 %	0.57 %
Tariq et al., (2021)	96.66 %	97.92 %	0.82 %	93.32 %	95.85 %	1.64 %	**99.99 %**	**100 %**	**0.01 %**	96.53 %	97.87 %	0.88 %

Table 9
Percent of feature reduction of the FARD-DFS methods in comparison with Mokbal et al., (2021).

Methods Name	1st Chunk	2nd Chunk	3rd Chunk	4th Chunk	5th Chunk	6th Chunk	7th Chunk	8th Chunk	9th Chunk
FARD-DFS-ACC	50 %	54 %	46 %	58 %	46 %	50 %	46 %	46 %	49 %
FARD-DFS-ALC	**53 %**	**59 %**	**49 %**	**65 %**	**62 %**	**51 %**	**64 %**	**51 %**	**56 %**
FARD-DFS-IM	50 %	53 %	40 %	59 %	43 %	40 %	60 %	50 %	38 %
FARD-DFS OA-OT	35 %	44 %	39 %	41 %	49 %	35 %	44 %	39 %	44 %
Mokbal et al., (2021)	The memory was reduced statically by 60 %								

Table 10
Percent of feature reduction of the FARD-DFS methods in comparison with Tariq et al., (2021) and Zhou & Wang, (2019).

Methods Name	1st Chunk	2nd Chunk	3rd Chunk	4th Chunk	5th Chunk	6th Chunk	7th Chunk	8th Chunk	9th Chunk
FARD-DFS-ACC	53 %	**67 %**	**57 %**	**60 %**	**53 %**	**63 %**	**47 %**	**33 %**	**67 %**
FARD-DFS-ALC	57 %	30 %	63 %	53 %	37 %	30 %	60 %	40 %	30 %
FARD-DFS-IM	40 %	57 %	53 %	27 %	27 %	50 %	23 %	60 %	47 %
FARD-DFS OA-OT	43 %	27 %	37 %	50 %	23 %	27 %	33 %	30 %	73 %
Tariq et al., (2021) Zhou & Wang, (2019)	No memory/feature reduction was implemented 0 %								



(a) D2-167 dataset

(b) D3-30 dataset

Fig. 6. Number of the selected features over time by using FARD-DFS methods.

approaches over both K. Liu et al., (2021) and Tariq et al., (2021) for almost all variants of our approach regarding both accuracy and F1-measure. In another observation, we found that all variants were superior to Tariq et al., (2021) with respect to accuracy with confidence levels of 0.0518, 0.0639, 0.0811, and 0.0445 of p-values for ACC, ALC, IM, and OA-OT, respectively. Furthermore, we find

Table 11
Overall accuracy results of the methods with respect to different feature size datasets.

Methods Name	D1-66 dataset		D2-167 dataset		D3-30 dataset		D4-30 dataset	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
FARD-DFS-ACC	99.42 %	99.82 %	97.06 %	98.27 %	**98.28 %**	99.00 %	98.29 %	**99.01 %**
FARD-DFS-ALC	99.06 %	99.82 %	96.64 %	98.67 %	98.00 %	99.50 %	96.39 %	98.74 %
FARD-DFS-IM	**99.59 %**	**100 %**	96.99 %	98.38 %	97.50 %	99.00 %	**98.37 %**	98.98 %
FARD-DFS OA-OT	99.52 %	99.91 %	**97.66 %**	**98.81 %**	97.83 %	99.50 %	97.98 %	99.06 %
K. Liu et al., (2021)	94.88 %	98.30 %	95.56 %	97.01 %	97.00 %	**100 %**	97.36 %	98.27 %
Tariq et al., (2021)	87.49 %	94.80 %	87.19 %	95.38 %	93.48 %	98.57 %	96.66 %	97.92 %

Table 12
t-test statistical significance testing of the accuracy and F1-measure results in comparison with benchmarks methods.

Methods Name	*t*-test			
	Accuracy		F1-measure	
	K. Liu et al., (2021)	Tariq et al., (2021)	K. Liu et al., (2021)	Tariq et al., (2021)
FARD-DFS-ACC	**0.0152**	**0.0518**	**0.0075**	**0.0847**
FARD-DFS-ALC	**0.0238**	**0.0639**	**0.0105**	0.1378
FARD-DFS-IM	0.1806	**0.0811**	**0.0726**	0.2335
FARD-DFS OA-OT	**0.0001**	**0.0445**	**0.0001**	**0.0581**

that each of ACC, IM, and OA-OT were superior over K. Liu et al., (2021) for both accuracy and F-measure with p-values of 0.0152, 0.0238 and 0.0001, respectively; and 0.0075, 0.0105, 0.0726 and 0.0001 for F1-measure of all suggested variants respectively. Secondly, we performed a Shapiro-Wilk test on the accuracy rates of the same dataset; the obtained p-values were 0.3944, 0.0037, 0.2524, 0.9999, 0.9115, and 0.002036 for the ACC, ALC, IM, OA-OT, K. Liu et al., (2021), and Tariq et al., (2021), respectively. This shows that the ALC and Tariq et al., (2021) are statistically more significant than other methods. Finally, in case the results are not normally distributed, we also applied the Friedman test as a non-parametric measure for calculating the statistical significance. Such testing method is based on a ranking mechanism instead of the average, since in some cases, the average is equal; thus, this method can handle this issue. The results of the Friedman test showed a significant efficiency of the accuracy of all our methods with compared approaches $R_j^2 = 21.89, p < .0001$. According to previous findings, we can conclude that the proposed approaches are statistically more significant performance than other compared methods.

6. Conclusion and future works

This article has provided a dynamic feature selection in the domain of XSS detection by considering incremental learning and enabling knowledge updates for both feature selection and classification simultaneously. The proposed method can be described as the first feature drift-aware XSS detection algorithm. It includes a DQN-MAFS as a novel framework for multi-agent dynamic feature selection using RL, including several components. The core contribution of the article is a sub-model for reward distribution named fair agent reward distribution based dynamic feature selection FARD-DFS. The latter provides four reward distribution techniques: accumulative contribution, alternative contribution, impurity based, and one action at one time. The proposed FARD-DFS approaches are compared with state-of-the-art algorithms; namely, the work of K. Liu et al., (2021) which also provides multi-agent based feature selection, and the work of Tariq et al., (2021) which provides and adaptive XSS detection using feature selection based on genetic and reinforcement learning. The results showed the superiority of FARD-DFS over the benchmarks in terms of the majority of metrics in general and with increasing the number of features in particular. Future work is to explore the incorporation of FARD-DFS with various types of XSS feature extraction in incremental learning. Another future work is to test our model in a federated learning scenario for XSS detection.

CRediT authorship contribution statement

Isam Kareem Thajeel Thajeel: Conceptualization, Methodology, Investigation, Writing – original draft. **Khairulmizam Samsudin:** Conceptualization, Investigation, Supervision, Writing – review & editing, Project administration. **Shaiful Jahari Hashim:** Writing – review & editing, Supervision. **Fazirulhisyam Hashim:** Writing – review & editing, Supervision.

Declaration of Competing Interest

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

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.

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.

Applebaum, S., Gaber, T., Ahmed, A., 2021. Signature-based and machine-learning-based web application firewalls: a short survey. Procedia Comput. Sci. 189, 359–367. https://doi.org/10.1016/j.procs.2021.05.105.

Barddal, J. P., Gomes, H. M., Enembreck, F., Pfahringer, B., & Bifet, A. (2016). On dynamic feature weighting for feature drifting data streams. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9852 LNAI*, 129–144. https://doi.org/10.1007/978-3-319-46227-1_9.

Barddal, J.P., Gomes, H.M., Enembreck, F., Pfahringer, B., 2017. A survey on feature drift adaptation: definition, benchmark, challenges and future directions. J. Syst. Softw. 127, 278–294. https://doi.org/10.1016/j.jss.2016.07.005.

Barddal, J.P., Enembreck, F., Gomes, H.M., Bifet, A., Pfahringer, B., 2019. Boosting decision stumps for dynamic feature selection on data streams. Inf. Syst. 83, 13–29. https://doi.org/10.1016/j.is.2019.02.003.

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

Caturano, F., Perrone, G., Romano, S.P., 2021. Discovering reflected cross-site scripting vulnerabilities using a multiobjective reinforcement learning environment. Comput. Secur. 103. https://doi.org/10.1016/j.cose.2021.102204.

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. Technol. Forecast. Soc. Chang. 168,. https://doi.org/10.1016/j.techfore.2021.120754 120754.

Cheng, Y., Komatsu, K., Sato, M., Kobayashi, H., 2021. A Deep reinforcement learning based feature selector. Commun. Comput. Inform. Sci. 1362, 378–389. https://doi.org/10.1007/978-981-16-0010-4_33.

Fan, W., Liu, K., Liu, H., Ge, Y., Xiong, H., & Fu, Y. (2020). *Interactive Reinforcement Learning for Feature Selection with Decision Tree in the Loop*. 1–12. https://doi.org/10.1109/tkde.2021.3102120.

Fan, W., Liu, K., Liu, H., Hariri, A., Dou, D., & Fu, Y. (2021). AutoGFS: Automated Group-based Feature Selection via Interactive Reinforcement Learning. *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, 342–350. https://doi.org/10.1137/1.9781611976700.39.

Fang, Y., Li, Y., Liu, L., Huang, C., 2018. DeepXSS: cross site scripting detection based on deep learning. ACM Int. Conf. Proc. Ser. 47–51. https://doi.org/10.1145/3194452.3194469.

Fang, Z., Wang, J., Geng, J., Kan, X., 2019. Feature selection for malware detection based on reinforcement learning. IEEE Access 7, 176177–176187. https://doi.org/10.1109/ACCESS.2019.2957429.

Fang, Y., Xu, Y., Jia, P., Huang, C., 2020. Providing email privacy by preventing webmail from loading malicious XSS payloads. Appl. Sci. (Switzerland) 10 (13). https://doi.org/10.3390/app10134425.

Ferone, A., Maratea, A., 2021. Adaptive quick reduct for feature drift detection. Algorithms 14 (2). https://doi.org/10.3390/a14020058.

Gronauer, S., Diepold, K., 2021. *Multi - agent deep reinforcement learning : a survey* (Issue 0123456789). Springer, Netherlands.

Gupta, C., Singh, R.K., Mohapatra, A.K., 2022. GeneMiner: a classification approach for detection of XSS attacks on web services. Comput. Intell. Neurosci. 2022 (1), 1–12. https://doi.org/10.1155/2022/3675821.

Heiderich, M., Schwenk, J., Frosch, T., Magazinius, J., & Yang, E. Z. (2013). mXSS attacks: Attacking well-secured web-applications by using innerHTML mutations. *Proceedings of the ACM Conference on Computer and Communications Security*, 777–788. https://doi.org/10.1145/2508859.2516723.

Huang, Y., Li, T., Zhang, L., Li, B., Liu, X., 2021. JSContana: malicious JavaScript detection using adaptable context analysis and key feature extraction. Comput. Secur. 104,. https://doi.org/10.1016/j.cose.2021.102218 102218.

Jim Manico, R. Rs. H. (2018). *XSS (Cross Site Scripting) Prevention Cheat Sheet - OWASP*. https://owasp.org/www-community/xss-filter-evasion-cheatsheet.

Kumar, J., Santhanavijayan, A., & Rajendran, B. (2022). Cross Site Scripting Attacks Classification using Convolutional Neural Network. *2022 International Conference on Computer Communication and Informatics, ICCCI 2022*. https://doi.org/10.1109/ICCCI54379.2022.9740836.

Kumar, S., Mallipeddi, R.R., 2022. Impact of cybersecurity on operations and supply chain management: emerging trends and future research directions. Prod. Oper. Manag. 31 (12), 4488–4500. https://doi.org/10.1111/poms.13859.

Li, X., Yao, J., Ren, J., & Wang, L. (2021). A New Feature Selection Algorithm Based on Deep Q-Network. *Chinese Control Conference, CCC, 2021-July*, 7100–7105. https://doi.org/10.23919/CCC52363.2021.9550745.

Liu, Z., Fang, Y., Huang, C., Han, J., 2021. GraphXSS : an efficient XSS payload detection approach based on graph convolutional network. Comput. Secur., 102597 https://doi.org/10.1016/j.cose.2021.102597.

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

Liu, K., Fu, Y., Wu, L., Li, X., Aggarwal, C., Xiong, H., 2021. Automated feature selection: a reinforcement learning perspective. IEEE Trans. Knowl. Data Eng. 4347 (c). https://doi.org/10.1109/TKDE.2021.3115477.

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.

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

Maurel, H., Vidal, S., Rezk, T., 2022. Statically identifying XSS using deep learning. Sci. Comput. Program., 102810 https://doi.org/10.1016/j.scico.2022.102810.

Melicher, W., & Fung, C. (2021). Towards a Lightweight , Hybrid Approach for Detecting DOM XSS Vulnerabilities with Machine Learning. *International World Wide Web Conference (WWW'21)*. 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.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *ArXiv*, 1–9. http://arxiv.org/abs/1312.5602

Mokbal, F.M.M., Dan, W., Imran, A., Jiuchuan, L., Akhtar, F., Xiaoxi, W., 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., Wang, D., Wang, X., Fu, L., 2020. Data augmentation-based conditional Wasserstein generative adversarial network-gradient penalty for XSS attack detection system. PeerJ Comput. Sci. 6, 1–20. https://doi.org/10.7717/peerj-cs.328.

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. J. Inform. Security Appl. 58,. https://doi.org/10.1016/j.jisa.2021.102813 102813.

Nguyen, T. T., Nguyen, N. D., Nahavandi, S., & Member, S. (2020). *Deep Reinforcement Learning for Multiagent Systems : A Review of Challenges , Solutions , and Applications*. *50*(9), 3826–3839

Nunan, A. E., Souto, E., Dos Santos, E. M., & Feitosa, E. (2012). Automatic classification of cross-site scripting in web pages using document-based and URL-based features. *Proceedings - IEEE Symposium on Computers and Communications*, 000702–000707. https://doi.org/10.1109/ISCC.2012.6249380

Paniri, M., Dowlatshahi, M. B., & Nezamabadi-pour, H. (2021). Ant-TD: Ant colony optimization plus temporal difference reinforcement learning for multi-label feature selection. *Swarm and Evolutionary Computation*, *64*(December 2019), 100892. https://doi.org/10.1016/j.swevo.2021.100892

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

Sahoo, D., Liu, C., & Hoi, S. C. H. (2019). Malicious URL Detection using Machine Learning: A Survey. *ArXiv*, *1*(1), 1–37. http://arxiv.org/abs/1701.07179

Sarmah, U., Bhattacharyya, D.K., Kalita, J.K., 2018. A survey of detection methods for XSS attacks. J. Netw. Comput. Appl. 118, 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.

Sato, D. M. V., De Freitas, S. C., Barddal, J. P., & Scalabrin, E. E. (2022). A Survey on Concept Drift in Process Mining. *ACM Computing Surveys*, *54*(9), 1–37. https://doi.org/10.1145/3472752.

Singh, A.K., 2020. Malicious and benign webpages dataset. Data Brief 32,. https://doi.org/10.1016/j.dib.2020.106304 106304.

Stock, B., Johns, M., Steffens, M., & Backes, M. (2017). How the web tangled itself: Uncovering the history of client-side web (in)security. *Proceedings of the 26th USENIX Security Symposium*, 971–987.

Sun, N., Zhang, J., Rimba, P., Gao, S., Zhang, L.Y., Xiang, Y., 2019. Data-driven cybersecurity incident prediction: a survey. IEEE Commun. Surv. Tutorials 21 (2), 1744–1772. https://doi.org/10.1109/COMST.2018.2885561.

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

Upadhyay, D., Sampalli, S., Plourde, B., 2020. Vulnerabilities' assessment and mitigation strategies for the small linux server, onion omega2. Electronics (Switzerland) 9 (6), 1–14. https://doi.org/10.3390/electronics9060967.

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

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. J. Parallel Distrib. Comput. 118, 100–106. https://doi.org/10.1016/j.jpdc.2017.07.006.

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. Comput. Secur. 113,. https://doi.org/10.1016/j.cose.2021.102554 102554.

Wu, Y., Li, M., Wang, J., Fang, Z., Zeng, Q., Yang, T., Cheng, L., 2022. Droidrl: reinforcement learning driven feature selection for android malware detection. SSRN Electron. J. 1–18. https://doi.org/10.2139/ssrn.4067267.

Xu, R., Li, M., Yang, Z., Yang, L., Qiao, K., Shang, Z., 2021. Dynamic feature selection algorithm based on Q-learning mechanism. Appl. Intell. https://doi.org/10.1007/s10489-021-02257-x.

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.

Ye, G., Tang, Z., Tan, S. H., Huang, S., Fang, D., Sun, X., Bian, L., Wang, H., & Wang, Z. (2021). Automated Conformance Testing for JavaScript Engines via Deep Compiler Fuzzing. *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 435–450. https://doi.org/10.1145/3453483.3454054.

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

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.