



Cross-site scripting adversarial attacks based on deep reinforcement learning: Evaluation and extension study^{*}

Samuele Pasini^{ID*}, Gianluca Maragliano^{ID}, Jinhan Kim^{ID}, Paolo Tonella^{ID}

Università della Svizzera italiana, Via G. Buffi 13, Lugano, 6900, Ticino, Switzerland

ARTICLE INFO

Editor: Professor Yan Cai

Keywords:

Cross-site scripting

XSS

Deep reinforcement learning

Adversarial attack

ABSTRACT

Cross-site scripting (XSS) poses a significant threat to web application security. While Deep Learning (DL) has shown remarkable success in detecting XSS attacks, it remains vulnerable to adversarial attacks due to the discontinuous nature of the mapping between the input (i.e., the attack) and the output (i.e., the prediction of the model whether an input is classified as XSS or benign). These adversarial attacks employ mutation-based strategies for different components of XSS attack vectors, allowing adversarial agents to iteratively select mutations to evade detection. Our work replicates a state-of-the-art XSS adversarial attack, highlighting threats to validity in the reference work and extending it towards a more effective evaluation strategy. Moreover, we introduce an XSS Oracle to mitigate these threats. The experimental results show that our approach achieves an escape rate above 96% when the threats to validity of the replicated technique are addressed.

1. Introduction

The proliferation of web applications has brought significant advancements but also introduced new security challenges. Among the various web-based attacks (Shahid et al., 2022; Marashdeh et al., 2021), Cross-site scripting (XSS) (Rodríguez et al., 2020) stands out as one of the most critical concerns. XSS attacks pose a significant threat as they can compromise user data, steal information, and spread worms. Malicious actors exploit vulnerabilities in web applications to inject harmful scripts, which are then unknowingly executed by users' browsers. To mitigate XSS attacks, robust detection methods and strong input validation techniques are essential to safeguard user data and system integrity.

Researchers have focused on the XSS vulnerability discovery, employing either static or dynamic analysis. Static analysis methods scrutinize the source code to identify potential attacks (Doupe et al., 2013; Steinhäuser and Gauthier, 2016; Mohammadi et al., 2017; Kronjee et al., 2018), but their application might not scale to the size of modern web applications or might result in overconservative results, with several false positives, due to the presence of programming constructs that are difficult to handle statically. Dynamic analysis, on the other hand, simulates user operations to detect attacks (Lekies et al., 2013; Stock et al., 2014; Fazzini et al., 2015). However, this approach suffers from a high false negative rate as test cases cannot cover all possible scenarios. To address these limitations, researchers have proposed methods to detect the injection of XSS scripts at runtime, complementing XSS vulnerabil-

ity discovery before release. In the early approaches, machine learning techniques with manual feature extraction were extensively used (Likarish et al., 2009; Nunan et al., 2012; Wang et al., 2014; Rathore et al., 2017; Mereani and Howe, 2018), followed by the advent of Deep Learning (DL) and the use of Deep Neural Networks (DNNs) for XSS detection (Fang et al., 2018; Mokbal et al., 2019; Tekerek, 2021).

While DNNs have shown great promise, they are vulnerable to adversarial attacks (Goodfellow et al., 2014a), where slight changes to input data can deceive the model. These attacks have successfully compromised DNNs used for XSS attack detection as well. A reference paper by Chen et al. (2022) proposed a Reinforcement Learning (RL) strategy to generate XSS adversarial examples and attack state-of-the-art (SOTA) XSS attack detectors based on DNNs. Their approach involves preprocessing, tokenization, and word vector representation using the Word2Vec model (Mikolov, 2013). The authors achieved almost perfect detection results (over 99% accuracy) and an impressive Escape Rate (ER)¹ of more than 90% against all DNN-based detectors.

However, we identified several threats to validity in the work by Chen et al. (2022). The first threat to validity is that the application of a sequence of actions could deteriorate the characteristics of the XSS script, and the authors did not apply any strategy to evaluate if the applied sequence of mutations is semantically preserving. The second

¹ The ER represents the percentage of adversarial examples that are not detected as malicious by the detector.

* Editor's note: Open Science material was validated by the Journal of Systems and Software Open Science Board.

* Corresponding author.

E-mail addresses: samuele.pasini@usi.ch (S. Pasini), gianluca.maragliano@usi.ch (G. Maragliano), jinhan.kim@usi.ch (J. Kim), paolo.tonella@usi.ch (P. Tonella).

threat is that the preprocessing pipeline of the detectors does not consider any potentially adversarial example, such that a mutation may potentially result in an out-of-vocabulary token (OOV) that is replaced by 'None' in the word vector representation. As a consequence, the input to the detector is no longer recognized as an XSS. The last threat concerns the lack of availability of different parts of the reference work, leading to a difficult replication, evaluation, and comparison.

In this paper, we replicate [Chen et al. \(2022\)](#) using a publicly available dataset and introduce an Oracle for XSS to test the occurrence of the hypothesized threats to validity. We extend the approach towards a more effective strategy by integrating the Oracle into the training process, addressing the identified threats while preserving the effectiveness of the original method. The proposed adversarial agent achieved a performance comparable to the reference work (less than 2% worse) while completely removing the threats to validity (more than 90% mitigation), demonstrating a more transparent evaluation strategy and a more effective training strategy.

The technical contributions of this paper are as follows:

- We replicate a reference work on Deep Reinforcement Learning for XSS adversarial attacks, using publicly available data and the public release of results.
- We identify the threats to the validity of the reference work and propose a method to mitigate them.
- We extend the reference work towards a more effective evaluation strategy by introducing an XSS Oracle and integrating it into the training process, effectively addressing the identified threats to validity.

The rest of the paper is organized as follows. [Section 2](#) explores the background related to XSS, Reinforcement Learning (RL), and XSS adversarial approaches, which are needed as preliminaries to understand the reference work. [Section 3](#) analyzes the reference work, with a focus on the possible threats to validity. [Section 4](#) presents the proposed method, focusing on the usage of an XSS Oracle and its integration into the reference work. [Section 5](#) describes the research questions, the experimental setting, and the process followed to replicate and extend the reference work. [Section 7](#) analyzes the results, [Section 8](#) describes the threats-to-validity of our work, while [Section 10](#) concludes the paper.

2. Background and related work

2.1. Cross-Site scripting (XSS)

Cross-Site Scripting (XSS) consists of the injection of malicious code into a web page. When a user visits the page, their browser unknowingly executes this script, leading to critical security breaches. It has been recognized as one of the prevalent threats, evidenced by the Open Web Application Security Project (OWASP),² a renowned authority on web application security, that has consistently ranked XSS as one of the top ten web application security risks. These attacks can have various malicious intentions, such as stealing confidential information or impersonating users to perform unauthorized actions. There are three primary types of XSS attacks, each with its own characteristics:

1. *Stored XSS* is the most severe form, where the malicious script is stored permanently in the server-side database. Any user accessing the affected page risks executing this script, potentially affecting multiple users.
2. *Reflected XSS* is a more targeted attack. The attacker lures the victim into visiting a malicious URL, often through spam emails. The URL contains harmful code, which is then executed in the victim's browser. This type of XSS is temporary and affects a specific user.

3. *DOM-based XSS* manipulates the Document Object Model (DOM) of a web page by modifying the input. This triggers the attack when the DOM is parsed on the client side, making it another non-persistent form of XSS.

2.1.1. Defending against XSS attacks

Given the diverse and harmful nature of XSS attacks, researchers have devoted efforts to developing effective defence strategies. The primary research focus has been on two key areas: XSS vulnerability discovery and XSS attack detection.

XSS vulnerability discovery: This encompasses both static and dynamic analysis techniques. *Static analysis* searches along all the possible execution paths in the code to find potential attacks. Several approaches have been proposed in this category. [Doupe et al. \(2013\)](#) suggested a server-side XSS mitigation strategy that isolates code from data, but this method falls short of dynamic JavaScript attacks. [Steinhauser and Gauthier \(2016\)](#) developed JSPChecker, a tool employing data flow analysis and string parsing to detect vulnerabilities in sanitization sequences. [Mohammadi et al. \(2017\)](#) utilized automated unit testing to identify vulnerabilities arising from improper input data handling. [Kronjee et al. \(2018\)](#) applied machine learning with a 79% precision rate to detect XSS and SQL injection vulnerabilities through static code analysis. *Dynamic analysis*, on the other hand, involves monitoring the data flow to pinpoint injection points and then testing for actual vulnerabilities. [Lekies et al. \(2013\)](#) introduced a technique to detect DOM-based XSS by monitoring and exploiting vulnerabilities in sensitive calls. [Fazzini et al. \(2015\)](#) automatically implemented Content Security Policies (CSP) in web applications, to track and manage the dynamic content.

XSS attack detection: While vulnerability discovery is essential, it may not offer complete protection against XSS attacks. Hence, researchers have also developed methods to identify malicious user input at runtime. This task is challenging due to the obfuscation techniques employed by attackers. Consequently, many detection methods rely on ML and DL approaches. [Likarish et al. \(2009\)](#) used JavaScript features for detection, achieving 92% accuracy. [Nunan et al. \(2012\)](#) refined this approach, improving detection. [Mereani and Howe \(2018\)](#) extracted structural and behavioral features, reaching 99% accuracy. [Fang et al. \(2018\)](#) utilized Word2Vec and LSTM, achieving precision and recall of 99.5% and 98.7%, respectively. [Mokbal et al. \(2019\)](#) constructed a large dataset and developed a feature selection technique, attaining 99.32% accuracy and 98.35% recall. [Tekerek \(2021\)](#) employed a CNN, achieving 97.07% accuracy on a public dataset. Despite the impressive results, State-Of-The-Art (SOTA) approaches present vulnerabilities that can be exploited by attackers, among which vulnerabilities to adversarial attacks against ML/DL.

2.1.2. Adversarial attacks on XSS detectors

The recent emergence of DL has led to groundbreaking advancements in various fields, including XSS attack detection, where it has achieved SOTA performance. However, researchers have identified a critical issue: the susceptibility of these methods to adversarial attacks. These attacks have successfully evaded multiple DL models across different domains, underscoring the imperative to enhance the robustness of these models. In the adversarial ML domain, several techniques are considered, and many of them are gradient-based, since they try to build adversarial examples applying noise computed by backpropagation ([Goodfellow et al., 2014b](#)). Differently, in the context of XSS attack detectors, several studies have explored adversarial attacks and most of them are based on Reinforcement Learning. [Fang et al. \(2018\)](#) developed an XSS adversarial approach utilizing the Dueling Deep Q Networks algorithm, but its escape detection success rate remained below 10% due to a simplistic bypass strategy. [Zhang et al. \(2020\)](#) proposed an algorithm based on Monte Carlo Tree Search (MCTS) to generate XSS adversarial examples for training the detection model. However, this algorithm relied on limited escape strategies and exhibited high time complexity. [Wang et al. \(2022\)](#) introduced a method employing soft Q-learning, dividing

² <https://owasp.org/www-project-top-ten/>

the bypass process into HTML and JS stages, achieving an impressive 85% escape rate.

The reference work by [Chen et al. \(2022\)](#) stands out with its Deep Reinforcement Learning algorithm, leveraging a set of mutation rules as actions, resulting in near-perfect escape rates against various SOTA XSS attack detectors. This approach will be thoroughly analyzed in [Section 3](#).

Beyond adversarial studies in security domains such as XSS, recent work has examined adversarial attacks against neural code models, particularly pre-trained programming language models. Although these models achieve strong performance on tasks such as code classification and vulnerability prediction, multiple studies have shown them to be vulnerable to semantics-preserving adversarial transformations. [Yang et al. \(2022\)](#) showed that natural, semantics-preserving code transformations can significantly degrade model performance without altering program behavior. Subsequent work confirmed the vulnerability of neural code models to superficial structural changes ([Yefet et al., 2020](#); [Bielik and Vechev, 2020](#)). More recent approaches focus on discrete, token-level attacks that operate directly in the program space and enable effective black-box adversarial generation while preserving functional equivalence ([Gao et al., 2023](#); [Jha and Reddy, 2023](#)). These findings highlight limitations of gradient-based adversarial methods in structured domains, drawing parallels with adversarial attacks on security payloads such as XSS.

2.2. Reinforcement learning

Reinforcement Learning (RL) is a distinctive machine learning paradigm that aims to maximize long-term rewards by striking a balance between exploration and exploitation. Unlike supervised learning, RL does not rely on labeled input-output pairs. Instead, it models the learning process as the interaction between two key components: the agent and the environment. The environment is represented as a timed sequence of states, $S = (s_0, s_1, \dots)$. At any given time t , the agent observes a state s_t and selects an action a_t from the available action space $A = \{a_0, a_1, \dots\}$ according to a policy $\pi(a_t|s_t)$, which is either the same being learned during the agent's interactions with the environment (*on-policy* learning) or which is kept separate from the policy under training (*off-policy* learning). The chosen action triggers a state change, and the new state s_{t+1} is determined by a Markov decision process with probability transition matrix $P(s_{t+1}|s_t, a_t)$. Simultaneously, the agent receives a reward r_{t+1} . The agent's objective is to maximize the long-term reward $R = \sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in [0, 1]$ is a discount factor. This balance between immediate and future rewards is a hallmark of RL.

Several algorithms have been developed for RL, each with its unique characteristics. One widely adopted algorithm is Proximal Policy Optimization (PPO) ([Schulman et al., 2017](#)), an on-policy algorithm that alternates between data collection through environment interactions and optimization of a clipped surrogate objective function via stochastic gradient descent. The clipping mechanism stabilizes training by limiting the policy updates, preventing drastic changes. Deep Deterministic Policy Gradient (DDPG) ([Lillicrap et al., 2020](#)) is an off-policy algorithm where the agent learns a deterministic policy guided by a Q-value function critic, which estimates the value of the optimal policy. DDPG employs target actor and critic networks and an experience replay buffer to enhance stability and learning efficiency. Soft Actor-Critic (SAC) ([Haarnoja et al., 2018](#)), another off-policy algorithm, is based on the maximum entropy framework. SAC trains the actor to maximize both expected reward and entropy, encouraging broader exploration. This approach has been shown to improve learning speed compared to state-of-the-art methods optimizing the traditional RL objective function. DDPG and SAC are typically applied to continuous action spaces. In contrast, PPO is versatile, supporting both continuous and discrete action spaces. RL approaches are very useful in several domains ([Shakya et al., 2023](#)), including adversarial attack generation. The reference work by [Chen et al. \(2022\)](#) proposed to train an adversarial agent able to attack XSS detectors using RL.

Table 1
Tokenization rules.

Regular Expression	Object
(?x)[\w\.\.]+?\(Javascript function
"\w+?"	Content within double quotes
\w+?"	Content within single quotes
http://\w+	URLs
<\w+>	Opening tags
</\w+>	Termination tags
\b\w+=	Attributes
(?<=\)\S+(?=\))	Content within parentheses

3. Reference work

In this section, we introduce the reference work ([Chen et al., 2022](#)) as follows. We begin with an overview of the proposed method. Then, we delve into the experiments described in the reference work. Lastly, we describe the potential threats to validity we identified, which prompted this replication and extension study.

3.1. Proposed method

As depicted in [Fig. 1](#), the authors of the reference work introduce a two-stage method, encompassing detection and escape phases. The detection phase involves utilizing an XSS detector, where the input undergoes preprocessing before being fed into the detector. Preprocessing comprises several steps: Positive examples, containing XSS attacks, are de-obfuscated and converted to lowercase. The URL is standardized to 'http://', and special characters such as angular brackets in '
' are removed. Tokenization is then applied to the examples using the rules outlined in [Table 1](#).

Tokenization converts each input example into a sequence of tokens. The 10% most frequent tokens are chosen for the vocabulary, while the remaining tokens are replaced with 'None'. Subsequently, a WordVec model is trained, representing each token as a 32-dimensional vector. The length of each example is standardized to 200 words, discarding excess words and padding shorter examples with 0. The resulting vectors are then fed into the detection models, as shown in [Fig. 1](#).

The escape phase involves crafting adversarial examples using RL. The idea is to train an agent to generate these examples effectively. The action space is carefully defined as a set of modification operations that can be applied to a malicious example while preserving its inherent characteristics. [Table 2](#) provides a comprehensive list of these possible actions.

The state space includes the historical record of actions taken by the agent. During each step, the agent chooses an action, updates its state accordingly, and then submits the transformed examples to the detection model. The agent receives a reward of 10 if the examples successfully evade detection, and a penalty of -1 if they are detected. This process continues iteratively until the agent either successfully bypasses the detection mechanism or reaches the maximum allowed number of steps.

3.2. Experiments

The training set for XSS detection models contains around 90,000 examples, collected from XSSed ([KF, 2012](#)) and Alexa ([Cooper, 2020](#)). This dataset is not publicly available and we had no way to craft it. To solve this problem, we used a publicly available dataset, as discussed in [Section 4](#). The authors of the reference work used the same dataset as [Wang et al. \(2022\)](#) to train the adversarial model. They trained MLP, LSTM, and CNN as detectors, and they considered also two commercial XSS detection systems, named Safedog ([Safedog, 2020](#)) and XSSChop ([Chaitin, 2019](#)).

Several metrics were employed to assess the performance of the detectors: True Positive (TP) represents a correctly identified XSS example, False Positive (FP) indicates a benign example wrongly classified

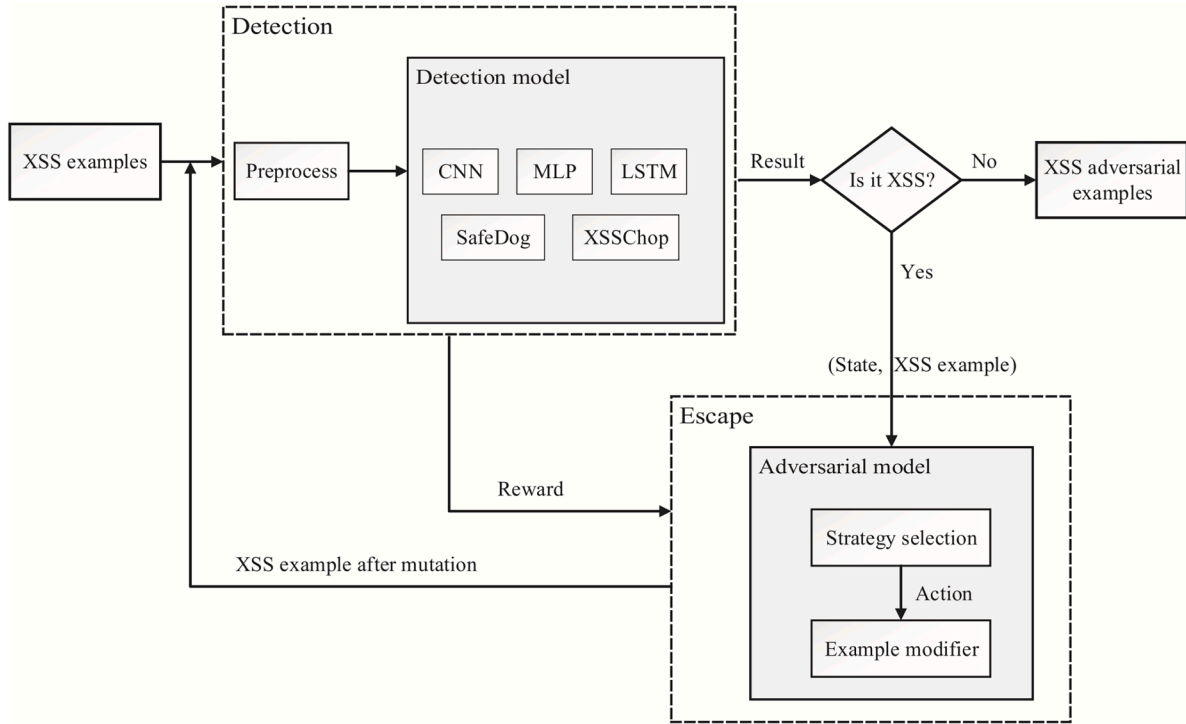


Fig. 1. Method proposed by the reference work (picture taken from the paper by Chen et al. (2022)).

Table 2

List of actions.

A1)	Add "" before "javascript"	A15)	Replace "(" and ")" with ""
A2)	Mixed case HTML attributes	A16)	Encode data protocol with Base64
A3)	Replace spaces with "/", "%0A" or "%0D"	A17)	Remove the quotation marks
A4)	Mixed case HTML tags	A18)	Unicode encoding for JS code
A5)	Remove the closing symbol of the single tags	A19)	HTML entity encoding for "javascript"
A6)	Add "
" to "javascript"	A20)	Replace ">" of single label with "<"
A7)	Add "	" to "javascript"	A21)	Replace "alert" with "top[al' + 'ert'](1)"
A8)	HTML entity encoding for JS code (hexadecimal)	A22)	Replace "alert" with "top[8680439.toString(30)](1)"
A9)	Double write HTML tags	A23)	Add interference string before the example
A10)	Replace "http://" with "//"	A24)	Add comment into tags
A11)	HTML entity encoding for JS code (decimal)	A25)	"vbscript" replaces "javascript"
A12)	Add ":" to "javascript"	A26)	Inject empty byte "%00" into tags
A13)	Add "	" to "javascript"	A27)	Replace "alert" with "top[/al/.source + /ert/. source](1)"
A14)	Add string "/drfv/" after the script tag		

as malicious, True Negative (TN) denotes a correctly identified benign example, and False Negative (FN) represents an XSS example wrongly classified as benign. Then, some derived metrics are defined as follows: *Accuracy* measures the proportion of correctly predicted examples (both malicious and benign) among the total. *Precision* calculates the ratio of correctly predicted malicious examples to all predicted malicious examples. *Recall* determines the ratio of correctly predicted malicious examples to all actual malicious examples. *F1-Score* is the geometric mean of Precision and Recall, aiming for high values of both.

When evaluating a adversarial attack, the authors focused on detection and escape rates. *Detection Rate* (DR) represents the ratio of malicious examples still detected by the XSS detection model, indicating the model's ability to defend against adversarial examples:

$$DR = \frac{\text{Number of malicious examples detected}}{\text{Total number of adversarial examples}} \quad (1)$$

Escape Rate (ER) refers to the percentage of malicious examples that go undetected and are recognized as benign by the detector:

$$ER = \frac{\text{Number of malicious examples undetected}}{\text{Total number of adversarial examples}} \quad (2)$$

The results presented in the reference work are summarized and analyzed in A

3.3. Identified threats to validity

The first threat to validity stems from the lack of validation of adversarial examples and their properties. The authors did not employ any strategy to ensure that the applied transformations preserve the semantic integrity of the examples. This omission raises concerns about the validity of the modified payloads. The second issue is related to the preprocessing and vocabulary construction. The initial dataset lacks tokens produced by the proposed payload transformations, or in some cases such tokens are rare in the dataset, meaning that new tokens resulting from the RL agent's actions are likely to fall outside the top 10% of considered tokens. Consequently, these tokens will be replaced with 'None', causing semantic changes that may invalidate the attack. This issue could lead to an inflated Escape Rate (ER) due to the disruption of the payload's semantics during preprocessing, making it challenging to assess the models' true detection capabilities. The final threat concerns the unavailability of crucial components of the reference work. The code, datasets, and models are not accessible, hindering further

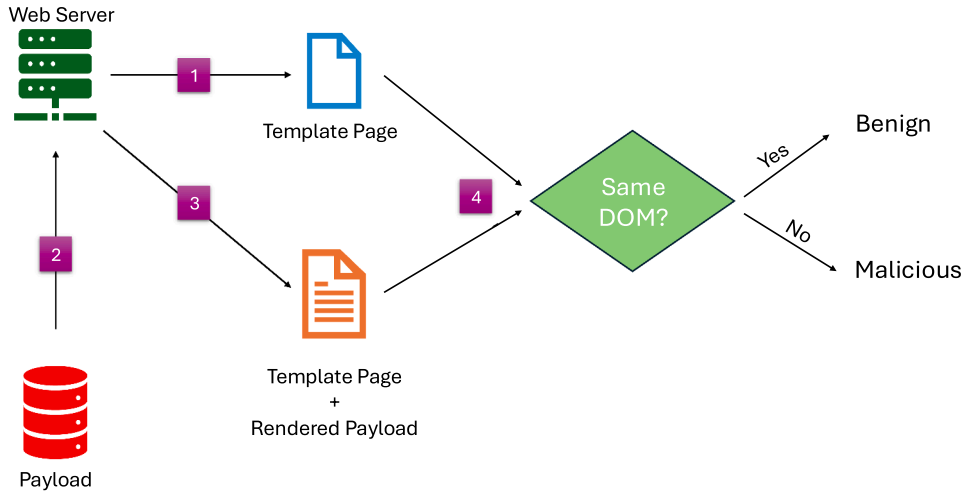


Fig. 2. Workflow of the XSS Oracle. A Payload is rendered in a known template of a Web Page and the DOMs of the two pages are compared.

analysis and replication of the reported results. This lack of transparency impedes the progress of research in this area, as it becomes difficult to build upon and extend the original findings. We have contacted the original authors asking them for code, datasets and models, but they never replied.

We summarize the three identified threats as follows:

- **TH1. Lack of validation of the actions:** The lack of validation for the applied actions raises questions about the semantic validity of modified payloads, potentially affecting the integrity of the examples.
- **TH2. Lack of validation of the preprocessed payload:** The preprocessing pipeline requires validation to ensure that the preprocessed payload maintains its semantic validity, which is crucial for accurate evaluation.
- **TH3. Lack of reproducibility:** The unavailability of experimental details, including code, datasets, and models, hinders reproducibility and limits the ability to extend and build upon the research, impeding further advancements in the field.

4. Methodology

The main idea of this paper is to introduce an XSS Oracle. As a first step, we demonstrate the Oracle's usefulness in assessing the validity of payloads and their potential impact. Furthermore, the Oracle can aid in developing a robust defense model, which allows an accurate evaluation of the performance of the approach proposed by Chen et al. (2022).

4.1. XSS Oracle

Based on the presence of an XSS attack inside of the payload of an HTTP request, we can consider two types of payload: 'Benign' and 'Malicious'. Benign payloads do not alter the DOM structure when executed, while malicious payloads cause changes in the DOM, potentially affecting the browser environment. As outlined in Fig. 2, we utilize the Oracle to mimic payload execution, observing the DOM of a template page rendered by a web server. The server accepts the payload as a parameter and incorporates its elements into the template. The Oracle then examines the DOM of the new page. If any differences are detected, the payload is labelled as Malicious; otherwise, it is classified as Benign.

4.2. Metrics

TH1 and TH2 arise from the agent's modifications and preprocessing of the payload, which could alter the characteristics of the XSS attack. To

address this, we employ an XSS Oracle that assesses the integrity of the attack properties, introducing the metric Ruin Rate (RR). Let us consider a set of payloads labeled as Malicious, denoted as $M = \{m_1, m_2, \dots\}$. This set M is generic and can include malicious samples from the original dataset or those generated by the XSS adversarial method.

How to structure the different sets for the evaluation of the different threats to validity will be discussed in Section 5.

We define a function $O(p)$ that, for any payload p , returns 1 if the Oracle classifies p as Malicious and 0 otherwise. For any set M , RR can be calculated as:

$$RR(M) = 1 - \frac{\sum_{m \in M} O(m)}{|M|} \quad (3)$$

If M contains samples from the original dataset, a non-zero $RR(M)$ indicates mislabeled examples. Conversely, if M consists of adversarial examples derived from an original set with $RR = 0$, a non-zero $RR(M)$ points to an adversarial process that has compromised the attack's properties.

TH1 and TH2 are also potentially related to an anomalous number of Out-Of-Vocabulary (OOV) tokens in the array fed to the detection model. We introduce a second metric, called OOV-Rate (OR), to evaluate this aspect. For an array of tokens V , $OR(V)$ is the number of the 'None' tokens present inside V (these represent OOV tokens) over the length of V :

$$OR(V) = \frac{\sum_{v \in V} OOV(v)}{|V|} \quad (4)$$

where function $OOV(v)$ is 1 if $v = None$, 0 otherwise.

5. Empirical study

This section presents the replication of the experiments in the reference work (Chen et al., 2022), highlighting the deviations from the results reported in the original paper. We introduce specific research questions for the replication and for the extension study, and we describe the experiments conducted to extend the reference work.

5.1. Replication study

In our replication study, we aim to closely follow the methodology of the reference work. However, some differences are worth noting and justifying. The dataset used to train the detectors was not publicly

Table 3

Dataset split between detector and adversarial agent, and then into train, validation and test sets.

Label	Detectors			Adversarial Agent		
	Train Set	Val. Set	Test Set	Train Set	Val. Set	Test Set
Benign	2884	721	902	0	0	0
Malicious	2883	721	901	2883	712	901

available, so we utilized an alternative dataset.³ This employed dataset is a well-known one (Mereani and Howe, 2018) containing more than 15,000 Malicious and Benign payloads. This dataset is built by collecting real-life attacks. The dataset for training the adversarial agent was partially available but had a different structure compared to the one employed in the reference work. The reference work does not adequately describe the correct payload structure, as the examples only consider parameters, while some steps mention filtering applied to the URL, suggesting the payload should be the entire HTTP request. Preliminary experiments revealed that datasets with varying structures encountered out-of-vocabulary issues even before the agent's actions were applied. In particular, when one dataset is used to create the vocabulary and to train a detector, and the other one is simply tested against it, RR is very high even before training an adversarial agent ($RR > 60\%$).

To isolate the identified threats to validity and mitigate any data structure-related problems, we divided the selected dataset into two parts: one for training the detectors and the other for training the adversarial agents, excluding Benign examples. This setup makes adversarial attacks more challenging, as the examples are closer to those used for detector training since they belong to the same distribution, thus they are more prone to presenting the same patterns. Consequently, a high RR in this context would indicate the significance of the threats TH1 and TH2, because of the alignment between detector's and adversarial agent's training sets.

The dataset was pre-filtered by the Oracle to ensure accurate labelling. After pre-filtering, the most representative class (Benign) was undersampled to ensure class balance. We computed the Ruin Rate of the original payload before and after preprocessing, resulting in a $RR = 0\%$, confirming that all samples were initially valid according to our Oracle. Furthermore, the dataset was split into train, validation, and test sets. Table 3 shows such details.

For the detection models, we used a different activation function in the output layer compared to the reference work. The softmax function used in the reference is more appropriate for multiclass classification, whereas our binary classification problem (Benign vs. Malicious) is better suited to the sigmoidal function. Furthermore, we employed only CNN, MLP, and LSTM as detectors, as Safedog and XSSChop are not publicly available. These models were trained for 150 epochs with early stopping (patience of 10 epochs), an embedding dimension of 8, a learning rate of 10^{-3} , and a stochastic gradient descent optimizer.

In contrast to the reference work, which used the SAC algorithm (Haarnoja et al., 2018) for agent training, we opted for the PPO algorithm (Schulman et al., 2017). This decision was made because we utilized the Stable Baselines library in Python for the Reinforcement Learning model implementation. The SAC algorithm implementation in this library is designed for a continuous action space, which does not align with our discrete action space, comprising discrete actions for mutating the attack payload. Therefore, we chose the PPO algorithm implementation, which handles a discrete action space.⁴

The performance of our detection models is similar to the reference work, achieving near-perfect metric scores as presented in Table 4.

Table 4

Performance of the XSS detection models of the replication study.

Detector	Precision	Recall	Accuracy	F1
MLP	99.67%	100.0%	99.83%	99.83%
LSTM	99.67%	100.0%	99.83%	99.83%
CNN	99.67%	100.0%	99.83%	99.83%

5.2. Research questions (RQs)

The first two RQs focus on evaluating the significance of TH1 and TH2:

- **RQ1. Evaluation of TH1:** Does the lack of validation of the actions pose a threat to validity?
- **RQ2. Evaluation of TH2:** Does the lack of validation of the preprocessed payload pose a threat to validity?

The final RQ extends this study by re-examining the performance of the method introduced in the reference work after addressing the identified threats to validity.

- **RQ3. Extension Study:** How does the reference method perform once the identified threats are mitigated?

5.3. Implementation

The details of the implementation were not available (see threat TH3). Hence, we proceeded with the development of a framework that we considered a good fitting of the requirements in the reference work, at the same time compatible with our own extension. Our experimental framework was implemented using Python 3.11. The DL library used to implement the models is PyTorch 2.2.1. The detectors are trained using a pre-processing pipeline that closely follows the instructions of the reference work, while the training procedure has been performed after a hyper-parameter tuning procedure. Fig. 3 shows the main steps performed for the implementation and the execution of the experiments of this study. The RL agent used for generating adversarial attacks is implemented in StableBaselines3 2.3.0.

The agent's actions are implemented following the instructions of the reference work (see Table 2 for the comprehensive list of actions that we have implemented), and have been manually evaluated and tested using the Oracle. The reward function $f(x)$ used by the RL agent was also taken from the original work (see Section 3.1), and is defined as follows for a mutated payload x :

$$f(x) = \begin{cases} 10 & \text{if attack } x \text{ goes undetected} \\ -1 & \text{if attack } x \text{ is detected} \end{cases} \quad (5)$$

The Oracle is implemented with a Web Server using FastAPI 0.104.0 and Jinja2 3.1.2 to render the template. The DOM is analyzed using BeautifulSoup 0.0.2 and zss 1.2.0.

5.4. Oracle integration and analysis

As shown in Fig. 4, the Oracle is used in two different stages. The set of undetected malicious payloads generated by the adversarial model, which represent the examples that contribute to the escape rate of the replication study, named E , is directly fed into the Oracle. The set E is then preprocessed as described in the previous sections, obtaining the set of arrays V . Also, V is fed into the Oracle. Thanks to the Oracle, it is possible to evaluate $RR(E)$ and $RR(V)$, which, respectively, represent the answers to RQ1 and RQ2. We do not rely only on the Oracle: the analysis of the Ruin Rate for RQ2 is complemented by the analysis of the Out-Of-Vocabulary Rate, that it is not reported in the Fig. 4 for simplicity. Regarding RQ3, there is no guarantee that increasing the vocabulary would solve TH2, since the adversarial agent is potentially able

³ <https://github.com/fmereani/Cross-Site-Scripting-XSS/blob/master/XSSDataSets/Payloads.csv>

⁴ <https://stable-baselines3.readthedocs.io/en/master/modules/sac.html>

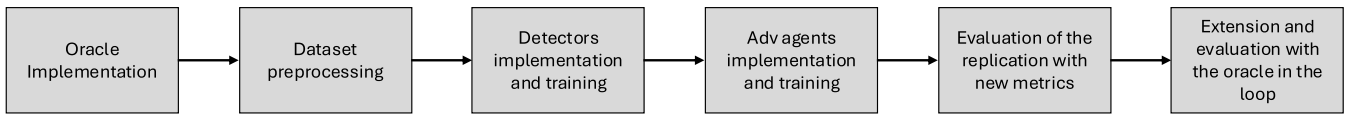


Fig. 3. Pipeline of the implementation and experiments, showing the main steps we performed for the replication and extensions study.

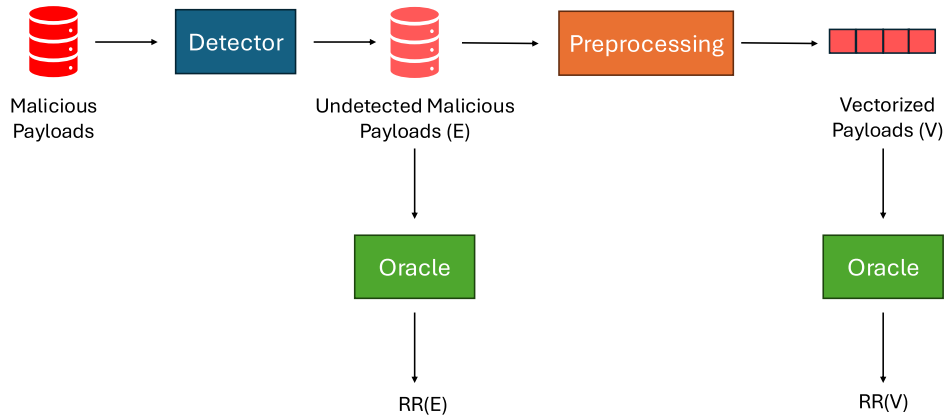


Fig. 4. Experimental setup: malicious undetected payloads are fed into the Oracle before and after preprocessing. The Oracle is used to assess whether Undetected Malicious Payloads (E) and their Vectorized representations (V) preserve the properties of an XSS attack, and to compute the corresponding Ruin Rate (RR).

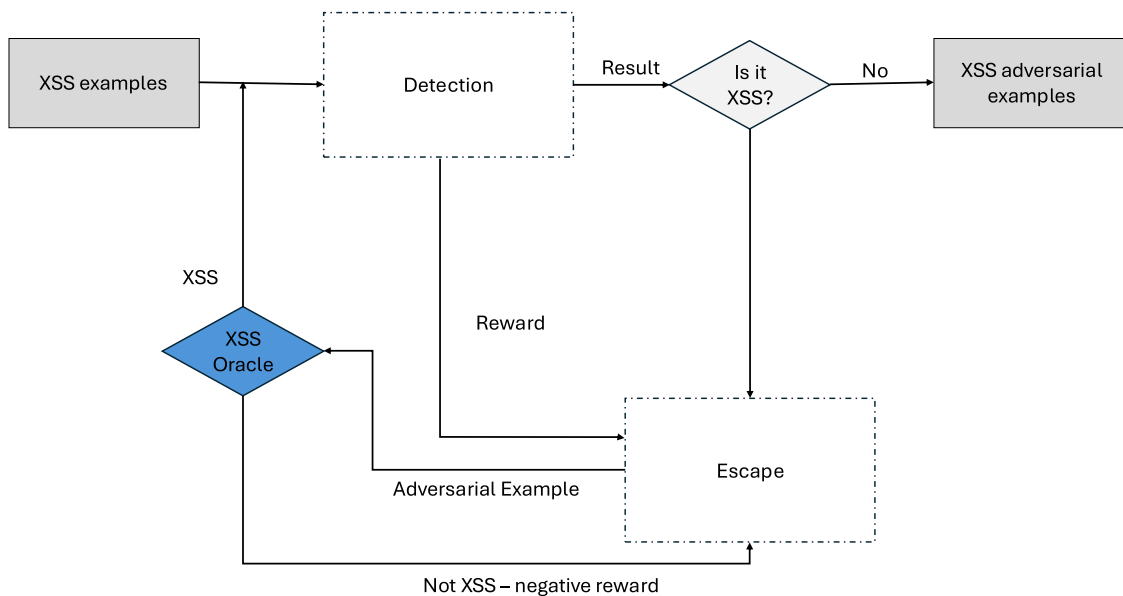


Fig. 5. Adversarial approach pipeline with the integration of the XSS Oracle. This is similar to the original one presented in Fig. 1, with the addition of the Oracle that returns a negative reward if the mutated attack is no longer a valid XSS.

to generate new tokens that are out-of-vocabulary regardless of the vocabulary size. To mitigate this threat-to-validity and to evaluate the real performance of the method, we integrated the Oracle into the training process of the adversarial agent. The agent's reward is set to -2 if the mutated payload, after preprocessing, is no longer recognized by the Oracle as an XSS attack. Otherwise, the reward is the same proposed in the reference work. The entire adversarial approach pipeline after the introduction of the Oracle in the loop is reported in Fig. 5.

6. Results of the replication study

Before analyzing the results, it is relevant to describe how we mitigate TH3 by replicating the results of Chen et al. (2022). We trained ten adversarial agents attacking each considered detection model, to deal with the non-determinism of the training process. Table 5 reports the average of the escape rates obtained by the adversarial agents. These

Table 5

Escape rate of the original adversarial agent, averaged across 10 training repetitions.

Detector	Escape Rate
LSTM	98.62%
MLP	99.73%
CNN	98.25%

ERs are almost perfect, demonstrating a consistency with the reference work, despite variations in the dataset and training algorithm (see Section 5.1). LSTM's ER is 6.58%pt⁵ higher than the reference work,

⁵ Percentage points %pt is the standard unit of measure for differences between percentages (e.g., 80% is 100%, or 40%pt, higher than 40%).

Table 6
Ruin rates of set E , averaged across 10 repetitions.

Detector	$RR(E)$
LSTM	6.34%
MLP	7.07%
CNN	6.36%

Table 7
Ruin and OOV rates of the array V , averaged across 10 repetitions.

Detector	$RR(V)$	$OR(V)$
LSTM	97.31%	47.49%
MLP	97.84%	44.76%
CNN	92.57%	43.85%

while the MLP and CNN results are identical and slightly lower (0.99%pt), respectively. This consistency strongly suggests that the differences in the dataset and training algorithm did not significantly impact the overall outcome. Our replication study successfully reproduced the reference work's results, confirming the effectiveness of the proposed adversarial agents.

7. Results

Since we successfully replicated the original study, with negligible differences in the results despite the changed dataset and training algorithm, we proceeded to test our hypotheses, trying to explain the reasons for such an amazing performance. We conjectured that the adversarial agents are exploiting vulnerabilities arising from the lack of validation of the actions and due to out of vocabulary tokens produced by preprocessing, which would rendering the other aspects of the algorithm less significant. In fact, any algorithm whose actions produce ineffective or out of vocabulary payloads would achieve a high escape rate, without generating any meaningful attack.

7.1. RQ1

(Evaluation of TH1)

We investigate the impact of the lack of action validation by analyzing the ruin rates of the set E , which contains all the generated payloads that successfully escaped detection. Table 6 presents the average ruin rates (RR_s) for each detection model.

Ruin rates are relatively low, ranging from 6.34% to 7.07%, indicating that the sequence of actions occasionally disrupts the semantics of the attack. However, this frequency is not high enough to be considered a significant threat to validity.

Answer to RQ1: The lack of validation of the actions is not a severe threat to validity, but it warrants further investigation to improve the detection model's robustness.

7.2. RQ2

(Evaluation of TH2)

For each adversarial agent, we collected all generated payloads that bypassed the detector, forming the set E . We then preprocessed this set to create the array V , and analyzed its ruin rate ($RR(V)$). The second column of Table 7 presents the average ruin rates across ten adversarial agents for each detection model. Ruin rates after preprocessing are notably high (exceeding 90% in all cases), indicating that preprocessing significantly disrupts the semantics of the XSS attack, posing a concrete threat to the validity of the original empirical study.

To find further explanations, we considered the content of V to assess whether our hypothesis about the introduction of 'None' (causing OOV

Table 8
Escape rates of the adversarial agent with the Oracle included in the training process, averaged across 10 repetitions.

Detector	Escape Rate
LSTM	98.13%
MLP	97.37%
CNN	96.89%

Table 9
Ruin and OOV rates of the array V , evaluated for RQ3 and averaged across 10 repetitions.

Detector	$RR(V)$	$OR(V)$
LSTM	0.01%	1.73%
MLP	0.11%	2.30%
CNN	0.08%	1.90%

tokens) could be a contributing factor to the performance of the original adversarial agent. The third column of Table 7 displays the average OOV rates across the ten adversarial agents for each detection model. OOV rates are significantly high, exceeding 40% in all cases, which is notably higher than the OOV rates of non-mutated payloads (around 6%). This confirms the threat to validity TH3 and suggests that preprocessing is one of the primary reasons for the agent's high escape rate, as adversarial agents learn to exploit OOV tokens as a *shortcut* to escape detection, rather than generating valid XSS payloads that can bypass detection while retaining their semantic integrity after preprocessing.

It is important to notice that the attack described in the reference work remains effective and poses a risk. However, in the original setup the preprocessing step makes the attack successful for any detection model. Any attack that trivially introduces OOV tokens is effective by construction in such setup. However, a defender aware of the OOV token issue would implement an additional layer of protection to discard payloads with an unusually high number of OOV tokens, rendering the attack ineffective, which complicates the assessment of the attack's true effectiveness against a wide range of detectors and raise questions about the vulnerabilities of commercial systems like XSSChop or SafeDog to such attacks, because all these detection systems might be in principle unaware of the OOV token problem.

Answer to RQ2: The lack of validation of the preprocessed payload poses a concrete threat to validity. This threat arises from the high ruin and OOV rates observed, indicating that preprocessing disrupts the XSS payload semantics. Adversarial agents exploit this, learning to bypass detection without preserving the payload meaning. This highlights the need for improved payload validation techniques to mitigate the possibility of attack shortcuts due to preprocessing.

7.3. RQ3 (Extension study)

In this RQ, the training process for adversarial agents closely mirrors that used in the replication study (Section 6), with a key difference: the Oracle is integrated to calculate a new reward function. The new reward is set to -2 if the mutated payload, after preprocessing, is no longer recognized by the Oracle as an XSS attack. Otherwise, the reward is the same proposed in the reference work. Table 8 reports the average escape rates achieved by the adversarial agent. Despite being very high, these rates are slightly lower than those obtained in the replication study (see Table 5).

The second and third columns of Table 9 report the average ruin rates (RR) and out-of-vocabulary (OOV) rates (OR) across the ten adversarial agents for each detection model. The low values of RR and OR indicate that the integration of the Oracle in the training process effectively mitigates TH3. These results demonstrate that it is feasible

to train adversarial agents capable of attacking XSS detection models as proposed in the reference work, without introducing any threats to validity related to preprocessing. In the new setup, the adversarial agent learns to produce payloads that include mostly valid tokens, while being still able to circumvent the detection capabilities of the considered detectors.

Answer to RQ3: Our Oracle-enhanced training method demonstrates the concrete threat of adversarial attacks on XSS detectors. Despite slight performance degradation, it confirms the ability of these attacks to bypass detection without exploiting threats related to the preprocessing.

8. Threats to validity

Internal validity. The training process of the adversarial agents is inherently non-deterministic. To ensure reliability of our findings, we repeated the training process for each agent ten times. For transparency and correctness of implementation, we have made our code publicly available, and we utilized well-known open-source frameworks for our implementation. The introduction of a customized XSS Oracle also poses a threat to internal validity, as it may be subject to misclassification and depend on the specific templates of the dataset used to construct and evaluate it.

External validity. While the employed dataset may not be exhaustive in representing every type of XSS attack, it is substantial and publicly accessible. Moreover, it has been widely used in previous research, establishing it as a suitable benchmark for evaluating XSS detection methods. Nevertheless, our results are inherently tied to the specific datasets and detector architectures considered, which may limit their generalizability.

Construct validity. We employed standard evaluation metrics in the security domain, including Precision, Recall, Accuracy, and F1-Score, to assess the detectors. For the adversarial agents, we used the escape rate as an evaluation metric, which aligns with the reference work. The Ruin Rate is defined with respect to a single, explicitly implemented XSS Oracle; different oracle designs or interpretations of attack preservation could therefore lead to different measurements.

Replication fidelity. Because the original implementation is not publicly available, our replication necessarily relied on a set of assumptions and exhibits differences from the reference work. In addition, the commercial XSS detectors used in the original study are not accessible. These factors introduce potential threats to replication fidelity.

9. Discussion

This work primarily focuses on identifying and mitigating the threats to validity present in prior research [Chen et al. \(2022\)](#). Our extension further leads to the design and adoption of a fairer evaluation framework - one that more accurately reflects the true capability of an attack algorithm to evade detection by a DNN-based detector. It is important to note, however, that an attack that remains undetected due to disruptive preprocessing still constitutes a successful attack, since it ultimately achieves the attacker's objective. Therefore, in this section, we discuss how our findings can inform practical defensive strategies.

A key takeaway from our study is the critical role of preprocessing and vocabulary construction. These components deserve as much attention and refinement as the DNN architecture itself, given their substantial impact on the detector's robustness. We recommend that detectors be systematically evaluated in adversarial setups, where metrics such as the OOV rate can guide defenders in identifying and prioritizing the weakest parts of their detection pipeline for improvement. In particular, the RR metric primarily exposes weaknesses in the design of the adversarial strategy, offering limited actionable insights for defenders. In contrast, the OOV metric is strongly tied to vulnerabilities in the preprocessing pipeline. Consequently, in adversarial testing scenarios, a

low RR coupled with a high OOV signals an urgent need to revisit and possibly redesign the preprocessing stage.

Finally, while our case study targeted a specific work ([Chen et al., 2022](#)), the same threats to validity are likely to affect other state-of-the-art approaches as well. Replicating additional SOTA studies using our proposed metrics would therefore provide a more comprehensive assessment of the robustness and generalizability of such detection systems.

10. Conclusion

In this paper, we replicated the study proposed by [Chen et al. \(2022\)](#) and conducted a thorough analysis of potential threats to its validity. After checking whether such potential threats actually affected the results reported in the original study, we presented an extended approach and introduced an extension study to mitigate them. Our findings are similar to those presented in [Chen et al. \(2022\)](#), but with a crucial difference: we eliminated the threats to their validity. This achievement allows us to propose a more effective method that directly attacks the detectors themselves, rather than relying on potential vulnerabilities in the preprocessing pipeline, associated with the generation of out of vocabulary tokens. Furthermore, our approach enhances transparency in the evaluation process, as we make code, datasets and results publicly available to all researchers in the field.

CRedit authorship contribution statement

Samuele Pasini: Writing – original draft, Validation, Supervision, Software, Methodology, Investigation, Conceptualization; **Gianluca Maragliano:** Software, Methodology, Investigation, Data curation; **Jinhan Kim:** Writing – review & editing, Writing – original draft, Methodology; **Paolo Tonella:** Writing – review & editing, Writing – original draft, Supervision.

Data availability

Data will be made available on request.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Paolo Tonella reports financial support was provided by European Union. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is funded by the European Union's [Horizon Europe](#) research and innovation programme under the project Sec4AI4Sec, grant agreement No [101120393](#).

Appendix A. Results reported in the reference work

The performance of the XSS detectors is reported in [Table A.1](#), showing the usefulness of the considered models in detecting XSS attacks with almost perfect results.

Regarding the adversarial attacks, [Chen et al. \(2022\)](#) computed the ER against all the detectors, showing almost perfect ERs, as reported in [Table A.2](#). These results were the starting point for our replication study, which was initially triggered by the astonishingly high performance exhibited by the proposed RL-based attack generator. We wanted to understand in depth the reasons for such amazing success.

Table A.1

Performance of the XSS detection models considered in the reference work.

Detector	Precision	Recall	Accuracy	F1
MLP	99.92%	98.00%	99.61%	98.96%
LSTM	99.97%	98.35%	99.65%	99.06%
CNN	99.85%	98.90%	99.76%	99.38%
XSSChop	99.61%	98.25%	99.14%	98.93%
SafeDog	100.00%	96.16%	98.47%	98.05%

Table A.2

Results of adversarial attacks in the reference work.

Detector	Escape Rate (ER)
MLP	99.73%
LSTM	92.04%
CNN	99.24%
XSSChop	98.46%
SafeDog	99.95%

References

- Bielik, P., Vechev, M., 2020. Adversarial robustness for code. In: *Proceedings of the 37th International Conference on Machine Learning*. JMLR.org.
- Chaitin, 2019. Xsschop: Xss detection engine. <https://xsschop.chaitin.cn/>.
- Chen, L., Tang, C., He, J., Zhao, H., Lan, X., Li, T., 2022. Xss adversarial example attacks based on deep reinforcement learning. *Comput. Secur.* 120, 102831. <https://doi.org/10.1016/j.cose.2022.102831>
- Cooper, 2020. Alexa. <https://www.alexa.com/>.
- Doupe, A., Cui, W., Jakubowski, M.H., Peinado, M., Kruegel, C., Vigna, G., 2013. Deducata: toward preventing server-side XSS via automatic code and data separation. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pp. 1205–1216.
- KF, 2012. Xssed: Xss attacks information. <http://www.xssed.com/archive>.
- Fang, Y., Li, Y., Liu, L., Huang, C., 2018. DeepXSS: cross site scripting detection based on deep learning. In: *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*, pp. 47–51.
- Fazzini, M., Saxena, P., Orso, A., 2015. AutoCSP: automatically retrofitting CSP to web applications. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 1, pp. 336–346. <https://doi.org/10.1109/ICSE.2015.53>
- Gao, F., Wang, Y., Wang, K., 2023. Discrete adversarial attack to models of code. *Proc. ACM Program. Lang.* 7 (PLDI). <https://doi.org/10.1145/3591227>
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014a. Generative adversarial nets. *Adv. Neural Inf. Process. Syst.* 27.
- Goodfellow, I.J., Shlens, J., Szegedy, C., 2014b. Explaining and harnessing adversarial examples. [arXiv:1412.6572](https://arxiv.org/abs/1412.6572).
- Haarnoja, T., Zhou, A., Abbeel, P., Levine, S., 2018. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: *International Conference on Machine Learning*. PMLR, pp. 1861–1870.
- Jha, A., Reddy, C.K., 2023. Codeattack: code-based adversarial attacks for pre-trained programming language models. In: *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI Press. <https://doi.org/10.1609/aaai.v37i12.26739>
- Kronjee, J., Hommersom, A., Vranken, H., 2018. Discovering software vulnerabilities using data-flow analysis and machine learning. In: *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pp. 1–10.
- Lekies, S., Stock, B., Johns, M., 2013. 25 Million flows later: large-scale detection of DOM-based XSS. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. Association for Computing Machinery, New York, NY, USA, p. 1193–1204. <https://doi.org/10.1145/2508859.2516703>
- Likarish, P., Jung, E., Jo, I., 2009. Obfuscated malicious javascript detection using classification techniques. In: *2009 4th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 47–54. <https://doi.org/10.1109/MALWARE.2009.5403020>
- Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N. M.O., Erez, T., Tassa, Y., Silver, D., Wierstra, D.P., 2020. Continuous control with deep reinforcement learning. *US Patent* 10,776,692.
- Marashdeh, Z., Suwais, K., Alia, M., 2021. A survey on SQL injection attack: detection and challenges. In: *2021 International Conference on Information Technology (ICIT)*, pp. 957–962. <https://doi.org/10.1109/ICIT52682.2021.9491117>
- Mereani, F.A., Howe, J.M., 2018. Detecting cross-site scripting attacks using machine learning. In: *Hassanien, A.E., Tolba, M.F., Elhoseny, M., Mostafa, M. (Eds.), The International Conference on Advanced Machine Learning Technologies and Applications (AMLTA2018)*. Springer International Publishing, Cham, pp. 200–210.
- Mikolov, T., 2013. Efficient estimation of word representations in vector space. [arXiv:1301.3781](https://arxiv.org/abs/1301.3781).
- Mohammadi, M., Chu, B., Lipford, H.R., 2017. Detecting cross-site scripting vulnerabilities through automated unit testing. In: *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 364–373. <https://doi.org/10.1109/QRS.2017.46>
- Mokbal, F. M.M., Dan, W., Imran, A., Jiuchuan, L., Akhtar, F., Xiaoxi, W., 2019. Mlpss: an integrated xss-based attack detection scheme in web applications using multilayer perceptron technique. *IEEE Access* 7, 100567–100580.
- 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. In: *2012 IEEE Symposium on Computers and Communications (ISCC)*, pp. 000702–000707. <https://doi.org/10.1109/ISCC.2012.6249380>
- Rathore, S., Sharma, P.K., Park, J.H., 2017. Xssclassifier: an efficient xss attack detection approach based on machine learning classifier on sns. *J. Inf. Process. Syst.* 13 (4), 1014–1028.
- Rodríguez, G.E., Torres, J.G., Flores, P., Benavides, D.E., 2020. Cross-site scripting (XSS) attacks and mitigation: a survey. *Comput. Netw.* 166, 106960. <https://doi.org/10.1016/j.comnet.2019.106960>
- Safedog, 2020. Safedog: web attack detection engine. <http://www.safedog.cn/>.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. [arXiv:1707.06347](https://arxiv.org/abs/1707.06347).
- Shahid, R., Marwat, S. N.K., Al-Fuqaha, A., Brahim, G.B., 2022. A study of XXE attacks prevention using XML parser configuration. In: *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*, pp. 830–835. <https://doi.org/10.1109/CICN56167.2022.10008276>
- Shakya, A.K., Pillai, G., Chakrabarty, S., 2023. Reinforcement learning algorithms: a brief survey. *Expert Syst. Appl.* 231, 120495. <https://doi.org/10.1016/j.eswa.2023.120495>
- Steinhauser, A., Gauthier, F., 2016. Jspchecker: static detection of context-sensitive cross-site scripting flaws in legacy web applications. In: *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*. Association for Computing Machinery, New York, NY, USA, p. 57–68. <https://doi.org/10.1145/2993600.2993606>
- Stock, B., Lekies, S., Mueller, T., Spiegel, P., Johns, M., 2014. Precise client-side protection against {DOM-based} {Cross-Site} scripting. In: *23rd USENIX Security Symposium (USENIX Security 14)*, pp. 655–670.
- Tekerek, A., 2021. A novel architecture for web-based attack detection using convolutional neural network. *Comput. Secur.* 100, 102096.
- Wang, Q., Yang, H., Wu, G., Choo, K.-K.R., Zhang, Z., Miao, G., Ren, Y., 2022. Black-box adversarial attacks on XSS attack detection model. *Comput. Secur.* 113, 102554.
- Wang, R., Jia, X., Li, Q., Zhang, S., 2014. Machine learning based cross-site scripting detection in online social network. In: *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on CyberSpace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICSS)*, pp. 823–826. <https://doi.org/10.1109/HPCC.2014.137>
- Yang, Z., Shi, J., He, J., Lo, D., 2022. Natural attack for pre-trained models of code. In: *Proceedings of the 44th International Conference on Software Engineering*. Association for Computing Machinery, New York, NY, USA, p. 1482–1493. <https://doi.org/10.1145/3510003.3510146>
- Yefet, N., Alon, U., Yahav, E., 2020. Adversarial examples for models of code. *Proc. ACM Program. Lang.* 4 (OOPSLA). <https://doi.org/10.1145/3428230>
- Zhang, X., Zhou, Y., Pei, S., Zhuge, J., Chen, J., 2020. Adversarial examples detection for XSS attacks based on generative adversarial networks. *IEEE Access* 8, 10989–10996.