How to Cite:
Mondal, B., Banerjee, A., & Gupta, S. (2022). XSS filter evasion using reinforcement learning to assist cross-site scripting testing. *International Journal of Health Sciences*, *6*(S2), 11779–11792. https://doi.org/10.53730/ijhs.v6nS2.8167

XSS filter evasion using reinforcement learning to assist cross-site scripting testing

Biswajit Mondal
Computer Science and Engineering, Dr. B C Roy Engineering College, Durgapur, 713206, West Bengal, India

Abhijit Banerjee
Electronics and Communication Engineering, Dr. B C Roy Engineering College, Durgapur, 713206, West Bengal, India

Subir Gupta
Computer Science and Engineering, Dr. B C Roy Engineering College, Durgapur, 713206, West Bengal, India
Corresponding author email: subir2276@gmail.com

***Abstract*---**Machine learning and deep learning are widely utilized and highly effective in attack classifiers. Little research has been undertaken on detecting and protecting cross-site scripting, leaving artificial intelligence systems susceptible to adversarial assaults (XSS). It is crucial to develop a mechanism for increasing the algorithm's resilience to assault. This study intends to utilize reinforcement learning to enhance XSS detection and adversarial combat attacks. Before mining the detection model's hostile inputs, the model's information is extracted using a reinforcement learning framework. Second, the detection technique is simultaneously trained using an adversarial strategy. Every cycle, the classification method is educated with freshly discovered harmful data. The proposed XSS model effectively mines destructive inputs missed by either black-box or white-box detection systems during the experimental phase. It is possible to train assault and detection models to enhance their capacity to protect themselves, leading to a lower rate of escape due to this training.

***Keywords*---**cyber security, reinforcement learning, machine learning, XSS.

Introduction

Web application firewalls manage HTTP traffic (WAF)[1][2]. Two of the most prevalent cyber attacks are cross-site scripting (XSS) and SQL injection (SQLI). XSS vulnerabilities were examined. Even on sites that appear to be safe, JavaScript may include potentially harmful code. This is known as cross-site scripting (XSS). In an XSS attack, one user passes malicious code to another. Code in a client-side script is used in cross-site scripting attacks[3][4]. Many online programs accept user input without verifying or encrypting it, leaving them vulnerable to attack. Unaware users may fall victim to a malicious script distributed using cross-site scripting (XSS). Because the browser cannot determine if the Script should be trusted, it runs it nonetheless, which may be dangerous. The malicious script may be able to access cookies, login information, or other sensitive data since the browser believes the feature came from a trustworthy source. Because the browser thinks the capability is dependable.

The Internet and online application services have exacerbated network security concerns. Cyberattacks destroy lives. SQL injection, file upload, XSS, and CSR are instances of online attacks. Cybercriminals want sensitive information or website administration. SQL injection, file upload, and other online vulnerabilities are prevalent. These are susceptible to XSS in browsers[5]. Consequently, the attacks pose a risk to user privacy and server security, exposing information and executing orders. Multiple research groups have integrated machine learning and deep learning into XSS detection. The evolution of attack detection technology has included adversarial AI-based assault strategies. An attacker evades detection models by creating aggressive and perplexing countermeasure samples to portray malevolent activities as innocuous. According to Generative Adversarial Networks, humans can still discern if a panda image is actual (GAN)[6]. One pixel can influence the classification outcome of a deep neural network (a "one-pixel assault"). The study of malware detection is also researched as a cyberattack.

It trains its detection system using samples of malicious software. Malware may be identified using reinforcement learning and GAN. There is insufficient research on how to utilize this to enhance XSS identification. Enhance the detection model's resistance to assault. Figure 1 demonstrates CSC. XSS makes use of reinforcement learning. By classifying preventative and mitigating data as XSS bad samples, we may be able to enhance the detection model's defense capability[7]. The following is a list of four main contributions:

- An XSS attack can be turned into an escape plan and the best plan because of the environment. Reinforcement learning is used to make this happen.
- Three XSS attack evasion strategies are proposed: encoding obfuscation, Hypersensitive word substitutions and morphological transformations.
- The detection model's ability to defend against adversarial assaults is continually improved.

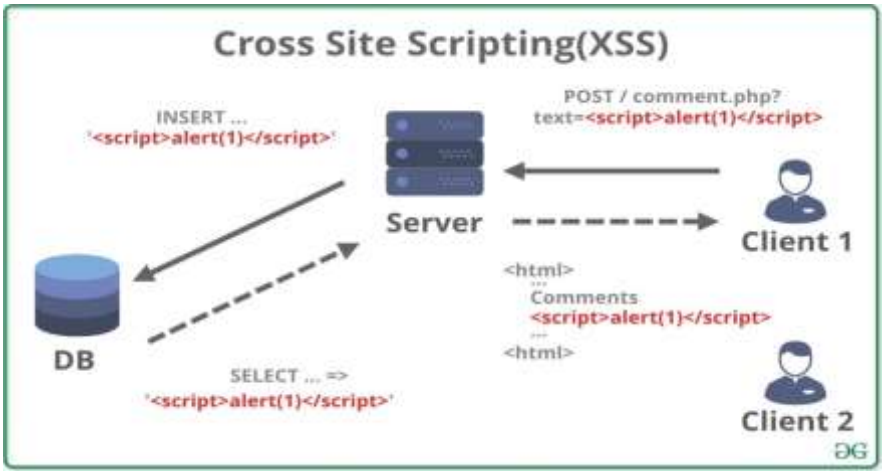

Figure 1: Cross-Site Scripting, https://www.geeksforgeeks.org/what-is-cross-site-scripting-xss

Literature Survey

Cross-site scripting and identification of vulnerabilities are crucial. Recent research combines these subjects. XSS detectors have been designed. Using their research, we enhance it.Ten machine learning techniques classify XSS and non-XSS attacks. The study of XSS produces attack vectors. Miscoded. Mohammadi developed a syntactic attack generator to assess cross-site scripting vulnerabilities. Researcher proposed a method for extracting XSS vulnerabilities using data from the most effective attack vectors. It may be used to build the XSS attacker library, resource library, and mutagens rule cat data for an attacker. Exploiting XSS vulnerabilities through attack vector design and automated testing is complicated by the diversity and complexity of Web applications. Classifying XSS receives less attention than identifying vulnerabilities or assaults. This study investigates how reinforcement learning may assist XSS in detecting attacks.

Machine Learning based XSS Classificatio Approaches

The Origin Policy limits the same Access to Netscape scripts. Only windows and documents made with JavaScript can be read and written. Insert Mozilla did not make any Script for Firefox, Sea Monkey, or any other browsers based on Mozilla. Any wrong input or Script will run if the web page is not cleaned up or encoded. Coding stops script tags from being opened and closed. OWASP's cheat sheet talks about XSS. Page integrity shows when a user gives false information. Tripwire checks the hash values of a website. If the value of the hash changes, the page is changed. Kirta and her coworkers made a defense against XSS on the client-side. They showed Boxes, a web proxy that can be controlled manually or automatically. It keeps XSS from leaking. Nunan et al. came up with a way to automatically group Cross-Site Scripting. We got information about how to classify things from URLs and websites. Web applications let users tell them what to do with files and databases. Shar and Tan say that what users type in can lead to security holes and attacks.

Experts think SQL injection and XSS vulnerabilities can be found through data mining. CFG SQL injection and XSS vulnerabilities for a critical time. Sensitive sinks can be put into groups using a data dependency tree and input cleaning techniques. An attack model for SQL injection and XSS was made with this feature vector. Later, testing became more efficient with the help of dynamic vectors and clustering[8][9].

Proposed Reinforcement Models for generation of XSS Adversarial Payload

This section talks about how to use reinforcement learning to get the adversarial Sample of these black-box or white-box XSS categorizations and improve the detection designer's ability to protect against attacks.

Feature Generation and Reward Function

Preprocessing has an impact on both black-box and white-box detection. This collection of dangerous samples is employed by the black-box detection tool. If the confidence level for detecting adversarial attacks is sufficiently high, detrimental samples are preserved and used for future white-box detection. Data sets with tainted samples can disclose XSS attacks.

Observation Space: Feature Encoding to byte/entropy histogram of Sample

2D Byte Histograms encode XSS samples. Malware is defined by 2D histogram entropy mapping. Low-dimensional feature extraction based on entropy. Encoded are feature vectors with 256 elements. Figure 2 shows the likely alert bytes histogram.

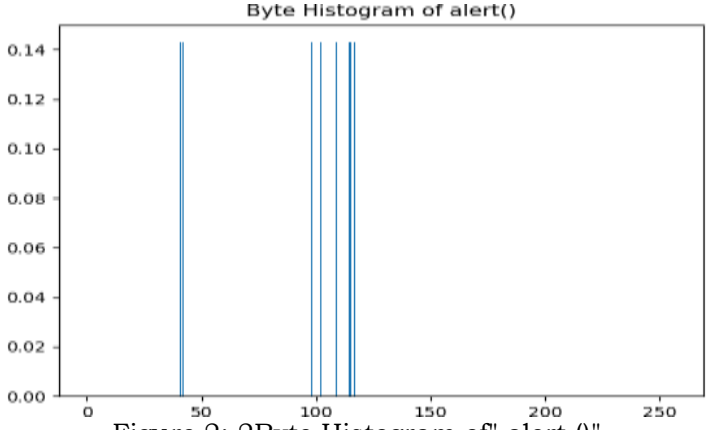

Figure 2: 2Byte Histogram of" alert ()"

Action Space: Adversarial Manipulation

Action space in RL is separate. RL does seven things. These different steps change the linguistic structure of the sample string without changing the way the XSS script is written (sample string). The adversarial Manipulation techniques or the

actions being taken by the RL algorithm the user to alter or change the XSS string structure are as follows:

- Char To 10: Converts a random character to Base10.
- Char To 16: Converts a random character to Base16[Hex]
- Char To 10 Zero: Converts a random character to Base 10[Decimal] and then
- Add Zero.
- Add Comment: Adds a random string of alphanumeric characters [a-z, A-Z] of length 10.
- Add Tab: Adds a random Tab[4spaces]to the String.
- Add Zero: Adds a Zero to a random position of the String.
- Add Enter: Adds an Enter [newline] character after a random location of the
- String.

Reward Shaping: Black box and White Testing

The module combines black-box and white-box APIs to provide an intuitive interface. "Post" is used to submit the identified Sample to the website using the REST API. XSS assaults are thwarted. When a request is approved or denied, a response is sent. Before feeding them into a white-box model, this interface processes samples. The confidence of XSS detection samples is assigned, and their results are communicated. Black and white boxes receive different rewards. The black-box approach encourages exploration. The reward of the white-box model depends on confidence[10][11]. Table 1 shows Examples of adversarial Manipulation on XSS string

$$r_t = \text{weight X Result}_{\text{value}} \backslash 1000 \qquad (1)$$

Example[with XSS JSON Payload]:

```
{
”name”:” <script>alert (’XSS Test’) </script>Bruce Banner”,
” description”:” <script>alert (’XSS Test’) </script>
Bruce Banner"
}
```

Expected Result $\text{Result}_{\text{value}}: 400$

Example [without XSS JSON Payload]:

```
{
" name":" Bruce",
" description": "Bruce"
}
```

Expected Result $\text{Result}_{\text{value}}: 200$

weight is taken as 10.

Table 1: Examples of adversarial Manipulation on XSS string

Sampled XSS String	Adversarial Method	Changed String
<h1/on	Char To 10	><h1/on

drag=confirm‘1‘)>Drag Me</h1>		drag=confirm‘1‘)>DragMe</h1>
<h1/on drag=confirm‘1‘)>Drag Me</h1>	Char To 16	><h1/on&#0x64;rag=confirm‘1‘)>Drag Me</h1>
<h1/on drag=confirm‘1‘)>Drag Me</h1>	Char To 10 Zero	><h1/on drag=confirm‘1‘)>Drag Me</h1>
<h1/on drag=confirm‘1‘)>Drag Me</h1>	Add Comment	><h1/o/*8888*/on drag=co/*8888*/nfirm‘1‘)>Drag Me</h1
<h1/on drag=confirm‘1‘)>Drag Me</h1>	Add Tab	>< h1/on drag=confirm‘1‘)>Drag Me</ h1>
<h1/on drag=confirm‘1‘)>Drag Me</h1>	Add Zero	><h1/on drag=confirm‘1‘)>Drag Me</h1>
<h1/on drag=confirm‘1‘)>Drag Me</h1>	Add Enter	><h1/on drag=confirm‘1‘)>Drag Me</h1>

Dataset

At the beginning of February 2007, XSS was made. It has information about cross-site scripting and is the largest online archive of websites that are vulnerable to XSS. In this analysis, XSSed was used in attacks from the other side and as a source of dangerous XSS samples. Over the last ten years, www.xsed.com has used real attacks. By retraining from this dataset, we can make examples ready for the real world.

Performance Metric

The SR (success rate) indicates the proportion of potentially hazardous samples deemed harmless by the target recognition model or instrument after escape modification. The more significant the fraction of escape vectors, the greater the likelihood of weakening the adversarial attack model.

$$SR = \frac{\text{Payload}_{\text{evaded}}}{\text{Payload}_{\text{evaded}} + \text{Payload}_{\text{detected}}} \text{ X } 100 \qquad (2)$$

Reinforcement Learning Algorithms Evaluated

Through reinforcement learning, good behavior is encouraged. A reinforcement learning agent can look at its surroundings, figure out what's going on, act, and learn from its mistakes[12][13]. With reinforcement learning, good actions are rewarded, and bad ones are punished. This method rewards good behavior and punishes bad behaviour. This tells the agent to think about what's best for the long run. Short-term goals can't be put off because of long-term goals. The agent has a happy attitude. AI that learns independently is controlled by penalties and

rewards (AI)[14][15]. RL challenges got better because of DQN policy gradient, TRPO, PPO, and evolutionary techniques (ES). Deep Q-learning methods come close to finding the best Q function for a state (DNNs). Policy gradients teach a DNN algorithm how likely it is that each state will behave differently. Open AI offers a simplified version of Natural Evolution Strategies that only know the mean. During training, ES was faster than DQN and A3C on challenging RL benchmark tasks, which led to better parallelization. Both techniques improve DNN parameters via stochastic gradient descent/ascent. DQN estimates loss gradient via backpropagation[16][17]. In the new system, policy gradients evaluate and confirm new ways of acting in a random way.

Deep Q Networks

In reinforcement learning, the use of function approximators has been shown to be very helpful. Most of the time, parameterized function approximators are used to show Q-functions.. $Q = \{Qw \mid w \in Rp\}$, where p is the number of parameters.

$$Q(s,a) := Q(s,a) + \alpha\left(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)\right) \quad (3)$$

where $\alpha \in R$ is the learning rate, s is the state, r is the reward. The update the equation is thus given as:

$$w := w + \alpha\left(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)\right)\nabla w Qw(s,a). \quad (4)$$

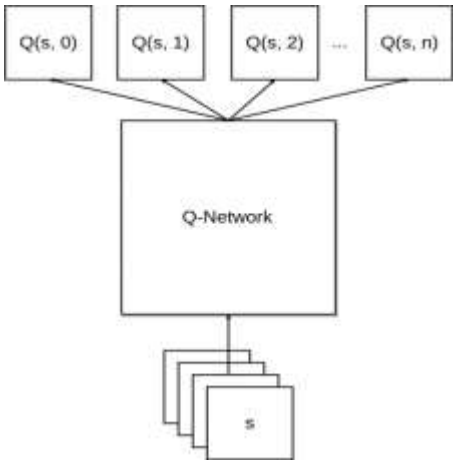

Figure 3: Deep Q-Network

Algorithm1 Deep Q- Learning with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with two random sets of weights θ, θ′
for episode = 1, M do
for t = 1, T do
Select a random action a_t with probability ε
Otherwise, select $a_t = argmax_a Q(s_t, a; \theta)$
Execute action a_t collect reward r_{t+1} and observe next state s_{t+1}

Store the transition $s_t, a_t, r_{t+1}, s_{t+1}$ in D
Sample mini-batch of transitions $s_j, a_j, r_{j+1}, s_{j+1}$
Set $y_j = r_{j+1}$ if s_{j+1} is terminal
$y_j = r_{j+1} + \gamma \max_{a'} Q(s_{j+1}, a';\ \theta'),$ otherwise
Perform a gradient descent step using targets y_j with respect to the online parameters θ
Every C step, set $\theta' \leftarrow \theta$
end for
end for

Double Dueling DQN

One Q-value estimate tells the other one what to do. We can get unbiased Q-value forecasts for the other estimator's activities by using unbiased estimators that are not biased[18]. Keeping updates separate from wrong assumptions helps get rid of bias. DDQN maintains two Q-networks:

- Online Network: Q_w terms $Q_w(s,a)$ and $\nabla_w Q_w\ (s,a)$ in update (4)
- Target network: update target in update (4)

The final update becomes:

$$w := w + a\left(r + \gamma \max_{a' \epsilon A} Q_{w^-}(s', a') - Q_w(s,a)\right)\nabla_w Q_w(s,a). \qquad (5)$$

The target Q-network is put on hold until an online value is given using function approximators based on deep learning. Online Q-network updates look unstable. It helps people learn. Even though Double Q-learning has two Q-networks, neither of them is a clear online or target network. Every time an update is made, the network used is decided by a 50/50 chance. It's important to note that these networks Q_{w1} and Q_{w2} play an equal role each time they are updated because their positions are decided stochastically with probability 0.5. The over-optimism of the goals in update (4) is reduced as a consequence.

Trust Region Policy Optimization: TRPO

TRPO changes policies to make them faster while ensuring that the new rules aren't too similar to the old ones. KL-Divergence, a measure of the distance between probability distributions, can be used to describe this constraint. Average policy gradients don't keep new policies close to old ones. Even small changes in size could affect how well an approach works. Large step sizes with plain slopes hurt the efficiency of sampling. When this happens, TRPO gracefully avoids it and steadily improves performance.

• TRPO is a policy-compliant algorithm.
• In both discrete and continuous action areas, TRPO can be applied.
• TRPO's Spinning Up implementation allows for MPI parallelism.

Let π_θ denote a policy with parameters θ. The theoretical TRPO update is:

$$\theta_{k+1} = argmax_\theta \, \& \, L(\theta_k, \theta) \qquad (6)$$

Where $L(\theta_k, \theta)$ is the surrogate advantage, denoting a metric π_θ explicitly showing how these fares with policy π_{θ_k} :

$$L(\theta_k, \theta) = E_{s,a\sim\pi_{\theta_k}}[\frac{\pi_\theta(a|s)}{\pi_{\theta k}(a|s)}]A^{\pi_{\theta_k}}(s,a) \qquad (7)$$

$D^-_{KL}(\theta, ||\theta_k)$ is the average KL-divergence among all policies visited by the old network.

$$D^-_{KL}(\theta, ||\theta_k) = E_{s\sim\pi_{\theta_k}}[D_{KL}(\pi_\theta(.|s)||\pi_{\theta_k}(.|s))] \qquad (8)$$

Results

For each RL method, we chose the following evaluation parameters:

- Number of Epochs/Episodes: 100
- Number of Runs per Epoch: 150
- Learning rate α for both networks: 0.002
- Greedy Parameter ϵ: 0.8
- Memory Stack: 2000
- Discount factor γ: 0.9

Network Topology for Online and Target network:

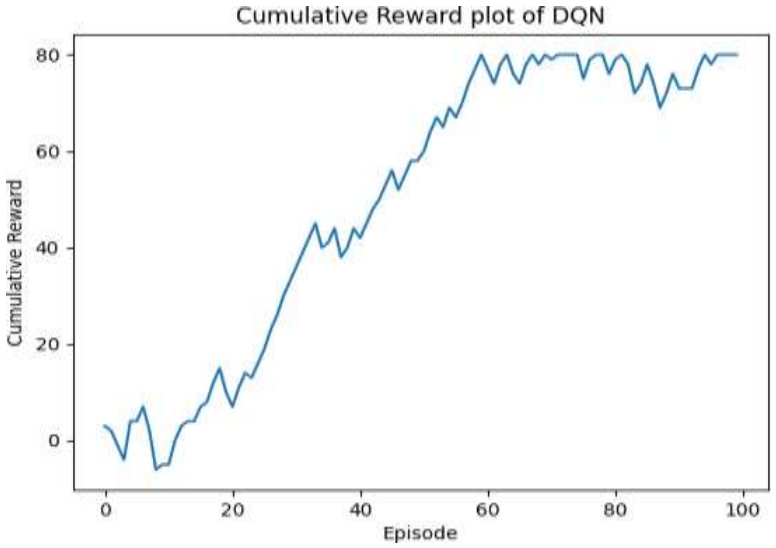

Figure 4: DQN Cumulative Reward

Net (
(fc1): Linear (in features=256, out features=50, bias=True)
(out): Linear (in features=50, out features=7, bias=True)
)

Cumulative Reward Plots

The following figures shows the cumulative rewards collected by each algorithm, in a total of 100 episodes. The cumulative rewards is also the success rate SR of evasion from the XSS detection method.
The cumulative reward of DQN is shown in Fig.4
The rewards for DQN are clipped at 80. After episode 50, the rewards have saturated. The Success rate Metric SR is thus:

$$SR = Result_{item}/(N_{TotalPayload} * 100 = 49.5\%) \quad (9)$$

The cumulative reward of DDQN is shown in Figure 5 .

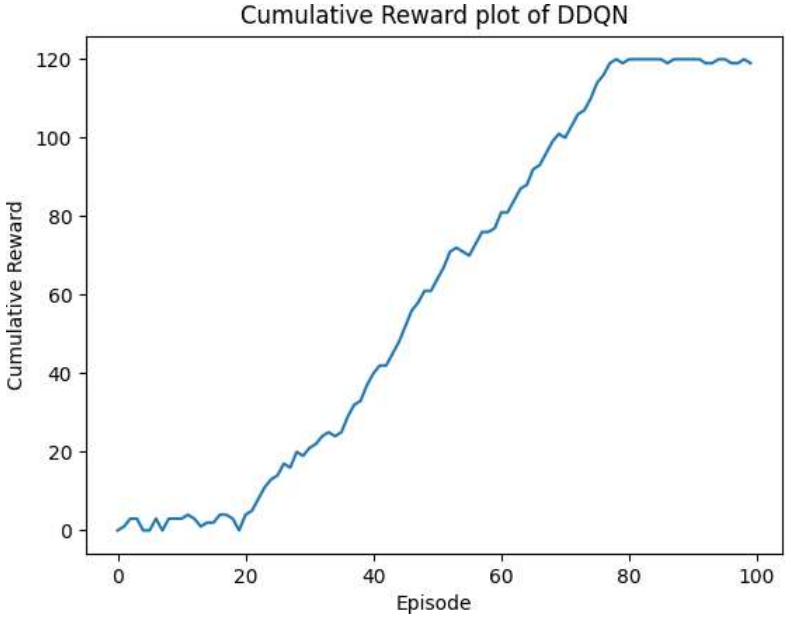

Figure 5: DDQN Cumulative Reward

The rewards for DDQN are clipped at 120. After episode 65, the rewards have saturated. The Success rate Metric SR is thus:

$$SR = Result_{item}/(N_{TotalPayload} * 100 = 60.75\%) \quad (10)$$

The cumulative reward of TRPO is shown in Fig.6.
The rewards for TRPO is clipped at 150. After episode 65, the rewards have saturated. The Success rate Metric SR is thus:

$$SR = Result_{item}/(N_{TotalPayload} * 100 = 85.7\%) \quad (11)$$

Comparative analysis of the RL algorithms on the basis of their success rate to evade the XSS detector is shown Fig.7. TRPO has the fastest convergence followed by SSQN and then by DQN. However, the success rate of DQN is

below 50%, which is more like coin toss. Therefore, it would be safe to say that, DDQn and TRPo would be the better RL method for XSS detection evasion with Adversarial Manipulation of XSS scripts. Table.3.1 shows the Test Sample, the actions taken and Manipulated Sample which resulted in the model to evade the XSS detection. These are the evasive obfuscated scripts which, is then added to the training list after every 10 episodes.

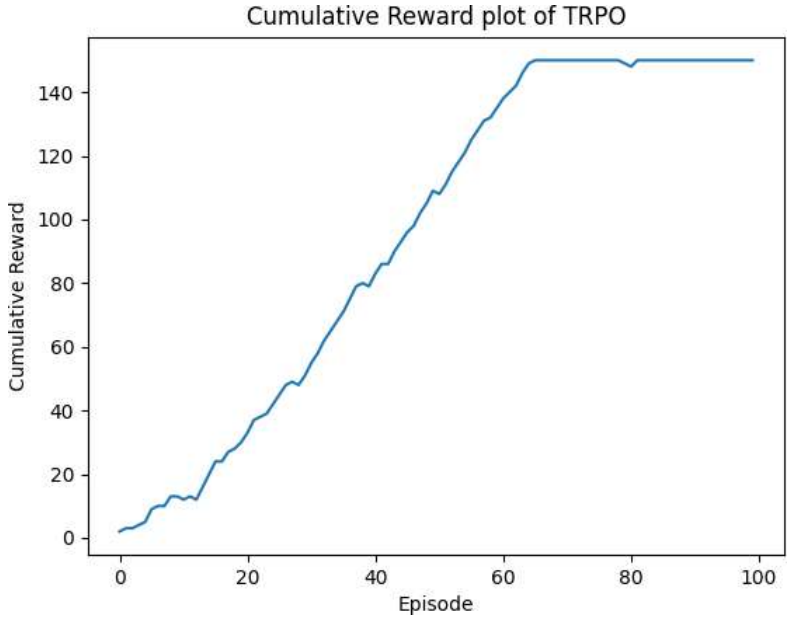

Figure 6: TRPO Cumulative Reward

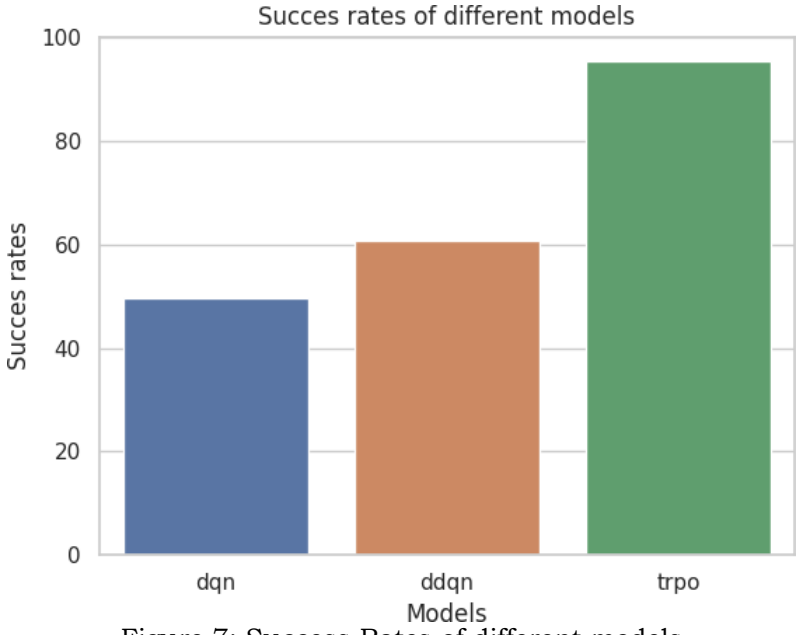

Figure 7: Success Rates of different models

Table 2: Adversarial Manipulation on XSS string, examples from Test Setwith the actions taken

Test Sample/Obfuscated Sample/Action
0 ORIGINAL: foo<script>alert (1) </script> OBFUSCATED: foo<sc/*8888*/ript>alert (1) </sc/*8888*/ript>ACTION: add Comment
1 ORIGINAL: foo<sc/*8888*/ript>alert (1) </sc/*8888*/ript> OBFUSCATED: foo<sc/*8888*//*8888*/ript>alert (1) </sc/*8888*//*8888*/ript>ACTION: add Comment
2 ORIGINAL: foo<sc/*8888*//*8888*/ript>alert (1) </sc/*8888*//*8888*/ript> OBFUSCATED:fo/*8888*/o/*8888*/<sc/*8888*//*8888*/ript>alert(1)</sc/*8888*//*8888*/ript>ACTION: add Comment
3 ORIGINAL: fo/*8888*/o/*8888*/<sc/*8888*//*8888*/ript>alert(1)</sc/*8888*//*8888*/ript>OBFUSCATED:fo/*8888*/o/*8888*/<sc/*8888*//*8888*/ript>a/*8888*/lert(1)</sc/*8888*//*8888*/ript>ACTION:addComment
4ORIGINAL:<inputtype=”text”value=“<div/onmouseover=’alert(1)’>X OBFUSCATED:<inputtype=”text”value=“<div/o/*8888*/nmo/*8888*/useo/*8888*/ver=’alert(1)’>X ACTION: add Comment

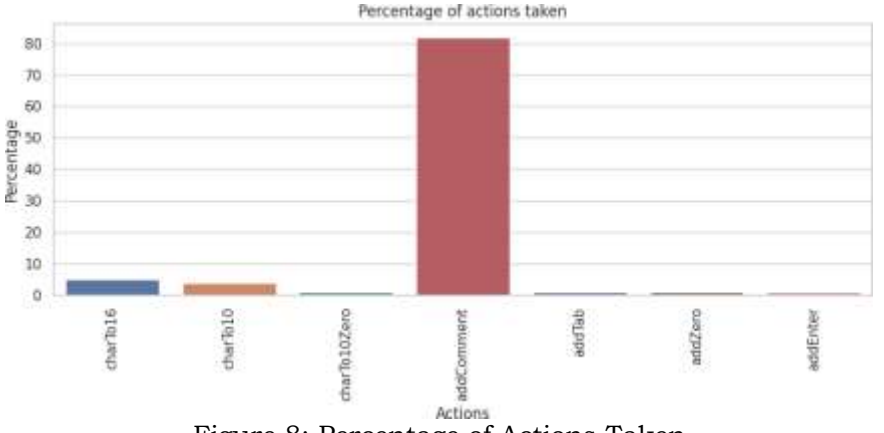

Figure 8: Percentage of Actions Taken

Java script comments suggest a greedy model. Statements trick XSS scanners. To confirm, compare all completed operations.. Figure 8 illustrates the percentage of total count for each activity. Commenting is the most prevalent RL activity, as shown in the graph. Some websites consider anything in a comment block to be safe and do not remove anything, so enabling the use of Cross-Site Scripting vectors. The system may also include anything in comment tags for user protection. Specific detection engines that operate by matching pairs of

open/close curly braces and then examining the label within may be able to thwart the XSS vector (before de-obfuscation). The double slash removes the unnecessary bracket, preventing a JavaScript error.

Conclusion

Cross-Site Scripting (XSS) security systems can be bypassed in several ways. These ways are called "XSS filter evasion" and are used by attackers. Hackers must first find a weakness in the program, then avoid input validation by the application and the server, and finally trick advanced browser filters to add harmful Script to client-side web page code. This article looks at some of the most common ways to get around XSS filters and explains how to reduce the risk of an adversarial attack in existing detection tools and models. Cross-site scripting, or XSS, is what this article is about. It looks at some of the most common ways to get around XSS filters. During this study, a Reinforcement Learning XSS adversarial attack method was made.

DQN, DDQN, and TRPO are the three methods that makeup RL. The rate of escaping at TRPO, 86%, was much higher than the rate at DQN (50 percent). Use Discrete Domain RL algorithms like TRPO to make XSS maneuvers that an attacker can't predict. By adding comments to the code, the RL model made it harder to understand XSS. The performance of the XSS classifier can be improved by retraining the model with adversarial data. In this study, avoiding an attack isn't as important as figuring out the best way to getaway. We mined malicious samples of both white-box and black-box detection models at the same time while keeping the attack function and avoiding being caught. We were able to hack the system it effectively. It retrained the detection model by classifying hostile sample data as "malevolent. ". By doing so, it maintained its outstanding detecting abilities and defended itself against outside threats. On top of that, the technique showed how both Burp Suite and Port Swigger's XSS escape mechanisms work. Soon, both SQL adversarial attack analysis and DDoS challenge and response architecture will use new technology.

References

1. A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Comput. Secur.*, vol. 31, no. 3, pp. 357–374, May 2012, doi: 10.1016/j.cose.2011.12.012.
2. Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman, "Identifying diverse usage behaviors of smartphone apps," in *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, 2011, pp. 329–344, doi: 10.1145/2068816.2068847.
3. B. Mondal, A. Banerjee, and S. Gupta, "review of SQLI detection strategies using machine learning," *Int. J. Health Sci. (Qassim).*, pp. 9663–9676, May 2022, doi: 10.53730/ijhs.v6nS2.7519.
4. B. Mondal, C. Koner, M. Chakraborty, and S. Gupta, "Detection and Investigation of DDoS Attacks in Network Traffic using Machine Learning Algorithms," *Int. J. Innov. Technol. Explor. Eng.*, vol. 11, no. 6, pp. 1–6, May 2022, doi: 10.35940/ijitee.F9862.0511622.

5. L. Erdődi, Å. Å. Sommervoll, and F. M. Zennaro, "Simulating SQL injection vulnerability exploitation using Q-learning reinforcement learning agents," *J. Inf. Secur. Appl.*, vol. 61, no. July, p. 102903, 2021, doi: 10.1016/j.jisa.2021.102903.
6. D. Chen, P. Wawrzynski, and Z. Lv, "Cyber security in smart cities: A review of deep learning-based applications and case studies," *Sustain. Cities Soc.*, vol. 66, p. 102655, Mar. 2021, doi: 10.1016/j.scs.2020.102655.
7. M. Baş Seyyar, F. Ö. Çatak, and E. Gül, "Detection of attack-targeted scans from the Apache HTTP Server access logs," *Appl. Comput. Informatics*, vol. 14, no. 1, pp. 28–36, 2018, doi: 10.1016/j.aci.2017.04.002.
8. H. Hanif, M. H. N. Md Nasir, M. F. Ab Razak, A. Firdaus, and N. B. Anuar, "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches," *J. Netw. Comput. Appl.*, vol. 179, no. August 2020, p. 103009, 2021, doi: 10.1016/j.jnca.2021.103009.
9. K. Natarajan and S. Subramani, "Generation of Sql-injection Free Secure Algorithm to Detect and Prevent Sql-Injection Attacks," *Procedia Technol.*, vol. 4, pp. 790–796, 2012, doi: 10.1016/j.protcy.2012.05.129.
10. H. Gao, S. Cheng, and W. Zhang, "GDroid: Android malware detection and classification with graph convolutional network," *Comput. Secur.*, vol. 106, Jul. 2021, doi: 10.1016/j.cose.2021.102264.
11. M. Breeding, "Current and future trends in information technologies for information units," *Prof. la Inf.*, vol. 21, no. 1, pp. 9–15, 2012, doi: 10.3145/epi.2012.ene.02.
12. S. Gupta, J. Sarkar, A. Banerjee, N. R. Bandyopadhyay, and S. Ganguly, "Grain Boundary Detection and Phase Segmentation of SEM Ferrite–Pearlite Microstructure Using SLIC and Skeletonization," *J. Inst. Eng. Ser. D*, vol. 100, no. 2, pp. 203–210, Oct. 2019, doi: 10.1007/s40033-019-00194-1.
13. S. Gupta, J. Sarkar, M. Kundu, N. R. Bandyopadhyay, and S. Ganguly, "Automatic recognition of SEM microstructure and phases of steel using LBP and random decision forest operator," *Measurement*, vol. 151, p. 107224, Feb. 2020, doi: 10.1016/j.measurement.2019.107224.
14. S. Gupta *et al.*, "Modelling the steel microstructure knowledge for in-silico recognition of phases using machine learning," *Mater. Chem. Phys.*, vol. 252, no. May, p. 123286, Sep. 2020, doi: 10.1016/j.matchemphys.2020.123286.
15. S. Gupta, "Chan-vese segmentation of SEM ferrite-pearlite microstructure and prediction of grain boundary," *Int. J. Innov. Technol. Explor. Eng.*, vol. 8, no. 10, pp. 1495–1498, 2019, doi: 10.35940/ijitee.A1024.0881019.
16. D. A. Linkens *et al.*, "Materials discovery and design using machine learning," *Comput. Mater. Sci.*, vol. 3, no. 3, pp. 1661–1668, 2016, doi: 10.1016/j.commatsci.2016.05.034.
17. S. Rao, A. K. Verma, and T. Bhatia, "A review on social spam detection: Challenges, open issues, and future directions," *Expert Systems with Applications*, vol. 186. 2021, doi: 10.1016/j.eswa.2021.115742.
18. A. Mchergui, T. Moulahi, and S. Zeadally, "Survey on Artificial Intelligence (AI) techniques for Vehicular Ad-hoc Networks (VANETs)," *Veh. Commun.*, vol. 1, p. 100403, 2021, doi: 10.1016/j.vehcom.2021.100403.