The ISC Int'l Journal of Information Security
ISeCure

July 2022, Volume 14, Number 2 (pp. 131–145)
http://www.isecure-journal.org

Bypassing Web Application Firewalls Using Deep Reinforcement Learning [**]

Mojtaba Hemmati [1], and Mohammad Ali Hadavi [1,*]
[1] *Faculty of Electrical and Computer Engineering, Malek Ashtar University of Technology, Tehran, Iran.*

A R T I C L E I N F O.

Article history:
Received: January 4, 2022
Revised: March 16, 2022
Accepted: April 13, 2022
Published Online: April 20, 2022

Keywords:
Adversarial Machine Learning, Reinforcement Learning, SQL Injection, Web Application Firewall (WAF)

Type: Research Article

doi: 10.22042/ISECURE.2022.323140.744

dor: 20.1001.1.20082045.2022.14.2.2.4

A B S T R A C T

Web application firewalls (WAFs) are used for protecting web applications from attacks such as SQL injection, cross-site request forgery, and cross-site scripting. As a result of the growing complexity of web attacks, WAFs need to be tested and updated on a regular basis. There are various tools and techniques to verify the correct performance of a WAF. But most of the techniques are manual or use brute-force attacks, so suffer from poor efficacy. In this work, we propose a solution based on Reinforcement Learning (RL) to discover malicious payloads, which are able to bypass WAFs. We provide an RL framework with an environment compatible with OpenAI gym toolset standards. The environment is employed for training agents to implement WAF circumvention tasks. The agent mutates the syntax of a malicious payload using a set of modification operators as actions, without changes to its semantic. Then, upon WAF's reaction to the payload, the environment ascertains a reward for the agent. Eventually, based on these rewards, the agent learns a suitable sequence of mutations for any malicious payload. The payloads, which bypass the WAF determine rules defects, which can be further used in rule tuning for rule-based WAFs. Also, it can enrich the machine learning-based WAFs datasets for retraining. We use Q-Learning, Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO) algorithms with the deep neural network. Our solution is successful in evading signature-based and machine learning-based WAFs. While our focus in this work is on SQL injection, the method can be simply extended to use for any string-based injection attacks.

1 Introduction

Despite lots of efforts on cybersecurity in the last decade, there are still vulnerabilities in the code, design, or operational environments of software products. Because of the vast attack surface and ease of exploitation, web attacks can cause intense damages. Web Application Firewall (WAF) is used to mitigate common web-based attacks such as cross-site scripting and SQL injection. WAF may come in the form of a server plugin, filter, or appliance [1]. A WAF is a type of reverse proxy that observes HTTP traffic to find out attack patterns based on its rule-set or poli-

[*] Corresponding author.
[**]This article is an extended/revised version of an ISCISC'18 paper.
Email addresses: hemmati@mut.ac.ir, hadavi@mut.ac.ir
ISSN: 2008-2045 © 2020 ISC. All rights reserved.

cies. Currently, there are two main types of WAFs, namely signature-based and anomaly based. In WAF based on anomaly detection, we can use machine learning tools [2]. Of course, there are also products with a hybrid approach. A common approach to setting a firewall rule-set is to use a blocklist. A blocklist contains string patterns of attacks, usually defined as regular expressions. Requests corresponding to the patterns are considered an attack and are blocked. There are various reasons a firewall does not work in the face of attacks, including problems in rule-set development, lack of rule-set tuning, misconfiguration, or implementation bugs. For example, a lack of sanity check for null bytes when parsing a JSON file is an implementation bug, because it allows the attacker to bypass the rules by injecting null bytes. Figure 1 shows an example of a modification of a payload that might circumvent a WAF. The second payload in this example tries to bypass the WAF filter by adding a URL encoded null byte %00 preceding a SQL keyword UNION. The second payload is syntactically different from the first one but has the same semantics. Preferably, a WAF must recognize both of them, but this might not happen in practice.

```
1.  'UNION  SELECT  password  FROM  Users  WHERE
    username-'admin'--
2.  %00'UNION SELECT password FROM Users WHERE
    username-'admin'--
```

Figure 1. Original malicious payload and modified payload

Diagnosing such problems requires going beyond the adversaries. Implementing an automated tool that measures a WAF's resistance to attacks is one way to ensure its performance.

In this work, extending [3], we propose a framework to test the correct performance of a WAF upon common web-based attacks. In particular, we employed this method on SQL injection payloads. In the first phase, an RL agent mutates a malicious payload and creates an identical payload with a different syntax. After sending a new malicious payload to an embedded WAF in an RL environment, a reward is determined for the agent based on the WAF response. In the training process, the agent learns an appropriate order of mutation operators for any type of payload. In our solution, we assume that the adversary does not have knowledge about WAF structure, so the agent sends a crafted malicious payload to a black-box WAF and tests its reaction to the payload.

This work due to its goal can be classified as an adversarial machine learning approach. Adversarial machine learning is a Machine Learning (ML) technique, which tries to mislead models by providing misleading input. The most common reason to use adversarial ML is to cause dysfunction in an ML model [4].

The major contributions of this work include:

- We extensively review, classify, and compare related works on adversarial approaches to circumvent ML-based security classifiers.
- We propose a framework to evade WAFs based on RL approaches that can be used for all types of WAFs.
- We develop an OpenAI Gym environment and manage experiments with Deep Q-learning (DQN), Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO) to bypass ModSecurity-CRS, Naxsi, and WAF-Brain using our framework. Due to large state spaces, we use the deep neural network as a function approximator.

In the following sections, we review related works in Section 2. Our method, experiments, and their results are expressed in Section 3, Section 4, and Section 5, respectively. Finally, we conclude the paper along with suggesting a couple of future works in Section 6.

2 Related Work

In this section, we review related works that use adversarial approaches for evading ML-based security classifiers in four subsections.

2.1 Adversarial ML in Intrusion Detection Systems

An Intrusion Detection System (IDS) is a software or device installed on network equipment to monitor policy violations or malicious activity and report audit analysis results. It uses two common diagnostic methods including anomaly-based and signature-based detection. The anomaly-based method reports any deviation from the normal behavior of the system and the signature-based method matches network traffic with the pattern of known attacks.

Caminero *et al.* [5] proposed an IDS based on the integration of two RL agents. They developed a simulated environment that acts as a second agent in an adversarial configuration against the original agent as a classifier. The environment agent first randomly extracts samples from a training dataset. The dataset contains examples of normal and abnormal behaviors and their labels. Then, it creates new attack samples and the classifier agent labels them using actions. The agent of the environment receives a reward or penalty according to the prediction. All positive rewards for the classifier agent are negative for the environment agent. Then, the environmental agent tries to increase the difficulty of prediction for the classifier agent by changing the samples. Both agents are trained in parallel using the DQN algorithm. They proved that their approach improves the performance of the classifier and reduces the prediction time.

Deokar and Hazarins [6] addressed the disadvantages of both types of IDSs. Anomaly-based IDS has the problem of high false alarm rates because it may recognize some unusual user activity as abnormal activity. Signature-based IDS cannot detect new types of attacks because of the use of a database, including known attack patterns. To solve these problems, they have proposed an IDS by collaborating between RL, Association Rule Learning, and Log Correlation techniques. RL rewards the system when selecting log files including anomalies and attack signs and fines it when selecting other log files. This procedure enables the system to select the correct log file when searching for attack traces.

Wu *et al.* [7] proposed a general anti-botnet traffic generator framework based on deep Q-learning. The RL agent can bypass the detector by adding perturbations to the botnet flow sample and changing its temporal and spatial properties. In this context, the attack takes place as a black box, and only a Boolean response is returned, which shows whether the current sample is identified. The authors used Auto-encoder to compress each flow sample into 1024 byte feature vector. They trained botnet detection models for evaluating their approach with CNN and decision tree algorithms. Using mutation functions such as modifying the timestamp of the first packet or appending a benign packet that is extracted from a normal flow, they can achieve evasion rates up to 50 percent on the CNN detection model.

2.2 Adversarial ML in Malware Detection Systems

Malware is a malicious code developed to harm computers and networks. The rapid growth in the number of malware attacks has increased the need for detection systems. The accuracy of these systems depends on the features extracted from the samples. These features are usually divided into static and dynamic categories. Static features are extracted directly from the code and dynamic features are extracted from the behavior of the program when it is running, including its access and communication patterns.

Anderson *et al.* [8] introduced a framework based on deep RL to circumvent ML-based malware detection systems. In this framework, an agent, without prior knowledge, attacks against the models. The authors claim that their approach is the first effective fully automated approach to deceiving malware detection systems based on changing the binary files. This framework facilitates the collection of disadvantages of a malware detection system and generates malicious examples for model retraining. The ACER algorithm is used to estimate the state-value. This technique helps the agent to learn effectively, even from a few experiences. The action space of this framework consists of operators that change the format of files while keeping the designated functionality of the malware. The environment comprises a series of malware samples and a malware detection model, the output of which specifies the state space and reward for the agent. The set of actions includes adding a specific function, manipulating names, deleting signatures, and so on. In each episode, the agent changes a sample of malware that the model can identify, to the extent that it can deceive the model. The policy learned by the RL agent can be generalized and used to change the new malware. The authors published an open-source OpenAI gym-based environment called gym-malware to train RL agents in this context.

Fang *et al.* [9] proposed a framework, which trains a deep convolutional Q-learning agent by constantly interacting with malware samples and applying a set of modifications to them. They implemented the MalwareEnv environment on the OpenAI gym toolset to fit into the malware detection evasion problem. For improving the learning pace, they used two hidden layers of Convolutional Neural Network with 256 and 64 filters, respectively. Their approach succeeds in evading an anti-malware classifier for about 75% of malware samples in backdoor types. This work is an improvement on Anderson *et al.*'s work [8], which uses the ACER algorithm in a similar RL environment.

2.3 Adversarial ML in Penetration Testing

Penetration testing is a controlled attack to examine the possibility of exploiting software vulnerabilities or misconfiguration in an organization's network. The use of automated solutions such as artificial intelligence compensates for the lack of experts in this field.

Pozdniakov *et al.* [10] proposed the Agent-Pen framework to solve the problem of automating the security audit by RL. Since the aim was to introduce a tool that can be used in different systems, RL algorithms have been used. The agent seeks to learn the optimal sequence of actions to perform penetration testing to achieve the maximum reward. This tool should be able to detect a sequence of attacks without the intervention of an expert. Because of the state space dimensions, a Recurrent Neural Network is used as an estimation function. Also, using the Auto-Encoder technique, the dimensions of the features that represent the state space are reduced and compressed into a smaller feature space. A sequence of attacks is created at the server-side and a reward signal is sent from the client-side. The authors claim that their approach can learn attack strategy and in addition, can use its knowledge to identify new targets.

Zennaro *et al.* [11] modeled penetration testing in the form of capture-the-flag challenges and apply the Tabular Q-learning algorithm to solve them. The authors designed five simplified CTF simulations with different scenarios to apply RL in penetration testing and analyzed the results. They highlighted two important challenges for an RL agent confronting CTF problems: the challenge of discovering the environment structure, and the challenge of learning using only inference and trying by different RL techniques such as lazy loading, state aggregation, and imitation learning. They addressed these challenges in their work.

Ghanem *et al.* [12] tried to introduce a new application of RL techniques in cyber security. They have developed a tool called IAPTS and defined the training environment for penetration testing agents with a Partially Observable Markov Decision Process. The starting point of this research is an automated penetration testing system that lacks efficiency and optimization and wastes many resources by conducting unsuccessful explorations. But at the endpoint, we have an automated penetration testing system with RL guidance that works well at both the training and testing levels. Such a system does not face problems such as human error and lack of time. The authors argue that RL capabilities can go beyond any penetration testing expert, especially in covering a variety of attack vectors, the time-consuming, and the accuracy and reliability of outputs.

Erdodi *et al.* [13] considered automating the process of exploiting the SQL injection vulnerability through RL algorithms such as tabular learning and deep Q-learning. They simplified that problem as a capture-the-flag challenge. An agent probes a system by sending queries, analyzing the answer, and exploiting a SQL injection vulnerability.

2.4 Adversarial ML in Web Application Firewalls

Demetrio *et al.* [14] presented WAF-A-MOLE, a tool that uses a set of mutation operators to craft malicious payloads in order to bypass ML-based WAFs. The methodology they used is guided mutational fuzz testing. It starts with a failing test and gets repeatedly transformed payloads through the random application of mutation operators. This process is repeated until a test is successful. The purpose of the test is to send a malicious SQL injection payload to the target WAF. A mutation operator just changes the syntax of a payload, while the semantics of the injected query is not affected. Therefore, the new payloads are the same as the original payloads in terms of maliciousness but reduce the degree of detection by WAF. This tool accepts an ML model as the target WAF, the initial malicious payload, and a threshold value as input. Then, creates a pool of payloads and prioritizes them according to the amount of confidence score the WAF has in recognizing them. Each payload is mutated until the confidence score is less than or equal to the initial threshold value. When a mutation operator is unable to generate a payload with a lower confidence score, the tool takes a step back and mutates on the previous payload with another operator. The bottleneck of such a system is its classification algorithm, so if a payload could not score better than its parent payload, it would be removed from the priority queue for better memory efficiency. The character-based, token-based, and graph-based methods were used to extract the features from payloads. The authors applied their approach upon some classifiers they had trained as well as upon WAF-Brain and SQLiGoT classifiers. The results showed that creating the payloads by selecting random operators in the input attribute space or fuzzing is not an effective approach. Thus, the WAF-A-MOLE tool is much more effective at generating malicious payloads with the classifier guidance and as expected, can bypass the WAFs examined in this paper. Their approach is gray-box and uses the confidence score assigned by the WAF.

Applet *et al.* [15] proposed an approach to test signature-based WAFs, called ML-Driven. Using ML, this tool learns attack patterns and creates a classifier to predict another substring of these attacks. The purpose of this technique is to optimally select a large set of attacks that can detect defects in the WAF. ML-Driven first generates random SQL injection attack payloads based on a designed grammar. Then, sends them to a web application protected by a WAF. The WAF labels the payloads with a pass or block. These results are used to construct a training set to train an estimation model to calculate the likelihood of bypassing the WAF for each payload. The authors have developed ML-Drive E (enhanced) with a combination of their previous work ML-Driven D (deep search) and ML-Driven B (broad search) [17]. In the enhanced version, the number of newly generated payloads is no longer fixed and is based on the probability of bypassing WAF for each of them. They applied their approach to ModSecurity and a private WAF. The results show that ML-Driven E works better than its predecessors and a random test strategy, as well as better than two vulnerability detection tools namely, WAF Testing Framework and sqlmap. This technique can provide better support for improving the rule-set in the WAFs.

Table 1. Summary of adversarial approaches for evading machine learning-based classifiers

Method	Objectives	Algorithms	States	Actions	Rewards
[5]	Using RL to learn two agents simultaneously	Deep Q-Learning	Samples of network packets	Dataset labels for classifier agent, Select the type of attacks in the next state for the environment agent	depends on the accuracy of the prediction
[6]	Addressing disadvantages in signature- and anomaly-based IDSs	RL, Rule Learning, Log Correlation	Log Files	Not specified	Not specified
[7]	Deceiving detection model by generate adversarial traffic flows	Deep Q-Learning	Botnet flow samples	Modifying the timestamp, Adding the length of the packet payload, etc.	0 when botnet flow was detected, +10 for evading model
[8]	Bypassing the ML-based malware detection system	Actor-Critic with Experience Replay	Features extracted from malware raw binary files	Adding a specific function, manipulation of method names, removing signature, etc.	+10 for system deception, and 0 otherwise
[9]	Bypassing malware detection system	Double Deep Q-Learning with Prioritized Experience Replay and Convolutional Neural Network	Features extracted from malware raw binary files	Add random values or library to the file, rename variables, etc.	0 for fail and positive reward for success
[10]	Automating the security audit by RL	Tabular Q-Learning, Deep Q-Learning with RNN and Auto-Encoder	A sequence of vulnerable ports	Scanning open ports for vulnerabilities, get access to the host,executing exploit code	Depends on the code maliciousness
[11]	Modeling penetration testing by RL	Tabular Q-Learning	Set of ports	Scanning open ports and their services, interacting with protocols, exploiting vulnerabilities, finding hidden files	-1 for every try and +100 for every access to the goal
[12]	Penetration testing of network infrastructure with RL	PERSEUS, Generalized Incremental Pruning, PEGASUS	operating system parameters, ports, service and programs, security extensions and connection records	Scan, fingerprinting, exploit code, probe, detect, connection, access, privilege escalation	depends on the value of the achievement and the time spent
[13]	Automating exploitation of SQL injection vulnerability through RL	Tabular learning Deep Q-learning	SQL responses	Identify the correct escape character, guessing the correct number of columns to insert in the SQL injection string	+10 for capturing the flag, -1 for other actions
[14]	Evading ML-based WAFs	Guided Mutational Fuzz Testing	Malicious SQL injection payload	Case swapping, white-space substitution, comment injection, comment rewriting, integer encoding, operator swapping, logical invariant	every payload is rewarded based on the WAF confidence score
[15]	Testing signature-based WAFs using ML	Evolutionary Search Algorithm	Malicious SQL injection payload	Selecting the payload with the highest probability of circumvention	Accept or reject
[16]	Evading WAFs using RL	DDQN, PPO	Malicious SQL injection payload	Space to comments, random case, swap keywords, swap integer base, swap white-space to alternatives, rewrite comment, change tautologies and etc.	+10 when reaching the goal and 0 for every try

Wang and Hu [16] proposed a similar approach to our work for evading the WAF problem. They trained a PPO agent for this purpose and get a success rate of 20% for evading WAF-Brain and 8.7% for ModSecurity over 1K payloads in evaluation. Based on their code [18] contrary to our work, they used the LibInjection library instead of a running version of ModSecurity. LibInjection [19] is an open-source library written in C programming language that detects SQL injection payloads using lexical analysis. This approach simplifies the problem, since ModSecurity [20] besides using LibInjection, is equipped with OWASP CoreRuleSet (CRS) [21]. CRS is a set of generic attack detection rules to use with ModSecurity or compatible WAFs. CRS was developed by the open-source community to increase the accuracy of detection and make it harder to evade. Our action set is also richer than their action set. Moreover, the authors in [16] stated that their token-histogram-based state representation may not be able to accurately estimate the state. However, we define our work state representation based on FastText embedding models. It is also noteworthy that their approach is based on the confidence score and considered as a gray-box method, while ours is considered as a black-box.

Table 1 compares related works based on the objectives, algorithms, definitions of state space, action space, and reward function. The reviewed works show that RL-based adversarial approaches have been used against various security classifiers as a successful solution. In the WAF evasion scope, there is room for improvements in action space, variety of testbeds, and more accurate representation of state space.

3 The Proposed Method

In this section, we discuss our main idea of using RL algorithms as a solution for the WAF evasion problem. We describe the model structure and RL environment characters which use for this goal. Mutation functions as action space, reward function, and agents are discussed further.

3.1 Concept

We formalize the WAF evasion problem using the Markov Decision Process (MDP). It is a discrete-time stochastic control process consisting of a 4-tuple *(S, A, T, R)* where:

- *S* is a finite set of states offered by the environment. In our problem, every payload represents a state.
- *A* is a finite set of actions to alter the payloads.
- *T(s, a): S × A →S'* is a transition operator which maps a current state, and any action taken in that state to a new state. In our problem, each payload after mutation with an action is transformed to a new state, as it is a new payload.
- *R(s, a, s'): S × A × S →R* is the reward function. It is the amount of reward the agent gets for taking an action *a* in state *s*, which leads to reaching state *s'*.

The goal of RL is to find a solution to an MDP. RL is a branch of ML approaches, which studies how an agent can interact with its environment to learn an optimal policy by trying to maximize cumulative rewards.

In this work, we focus on WAF evasion using SQL injection payloads since SQL injection is a common attack, mentioned in the OWASP top 10 as the third case [22]. It occurs because user-supplied data is not validated, filtered, or sanitized by the application. developing a specific action set, our framework can also be used for other web application attacks such as cross-site scripting.

3.2 Model Structure

The environment, agent, action space, and reward should be defined before using RL to address the WAF evasion problem. The environment consists of several sections, a server supported by the WAF, a pool of malicious payloads, and a feature extractor, which converts payloads into numeric vectors. Figure 2 demonstrates the structure of our method.

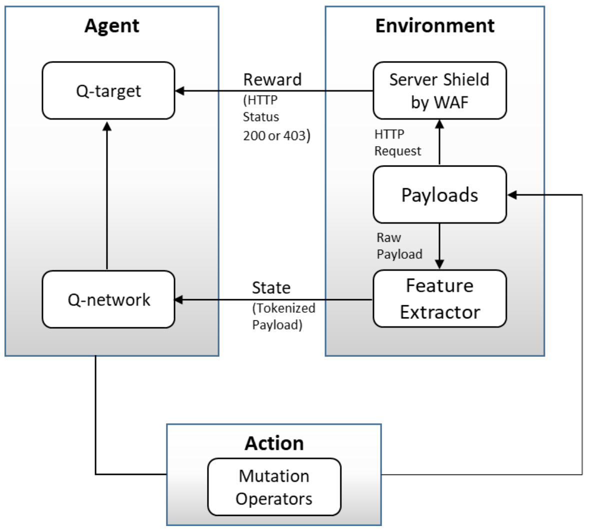

Figure 2. The structure of the proposed method

The agent receives a malicious payload as the current state, performs a mutation operation on it, and sends it as an HTTP request to the WAF. The WAF response to this request could be an HTTP status 200 (allowed) or 403 (forbidden). The response is used to determine the reward. The environment also returns a mutated payload as a next state. The building blocks of our method, shown in Figure 2, are explained in the following subsections.

3.3 Environment

In our framework, the environment includes the raw string malicious payloads that are sensitive to each action. A raw string payload feeds directly to a WAF. The training process starts with a payload, which has more features to mutate. After each episode, the number of features decreases and the evasion operation becomes more difficult. The payloads used in this work were extracted from the SQL injection payloads dataset from Demetrio *et al.* [14].

To be able to work with high-dimensional complex data such as text in neural networks, we have to first convert the text data into numerical vectors. Therefore, we train the FastText embedding model on the dataset and utilize it in the feature extractor unit (Figure 2). The embedding layer transforms each word into a fixed-length vector of a defined size. Fixing the length of word vectors leads to a better representation of words along with reduced dimensions. FastText embedding treats each word as a composition of character n-grams, so the vector for a word is made of the sum of this character n-grams. Another common technique, word2vec treats words as the smallest unit to train on. This means that FastText can generate better word embedding for rare words [23]. Each malicious payload after tokenization and feature extraction using these embedding models is converted to a two-dimensional array structure in which each row represents an embedding vector for every token of the malicious payload. After applying normalization, we define these arrays as the state observed by the agent.

Our framework can be utilized against all types of WAFs. However, in this work, we mainly focus on three open-source products, including ModSecurity-v3, Naxsi [24] , and WAF-Brain. Section 4.1 provides more details regarding the implementation of our framework.

3.4 Action Space

When the agent meets the current state of the environment, it must select an action from the actions set and deploy it on the state. An action is a mutation function to changes the syntactical shape of the payload while preserving its semantics. Below, we describe our action set:

(1) *Add comments*: adding comments between SQL clauses with random content.
(2) *Swapping spaces to comments*: substituting comments with random content for all white spaces in the payload.
(3) *Comment rewriting*: replacing the content of existing comments with random texts.
(4) *Swapping operators with their alternatives*: replacing some operators with alternatives, e.g., replacing "OR" with "——".
(5) *Swapping integer representation*: randomly replacing integers with other representations such as Hex or Base-64 encodings.
(6) *Swapping space to alternatives*: substituting white space with alternatives, e.g., replacing "\n" with "\t".
(7) *Swapping tautologies*: for example, replacing "OR 1=1" with "OR B=B".
(8) *MySQL executable comment*: This mutation is only applicable in MySQL. e.g., "SELECT" to "/*!50000SELECT*/".
(9) *Clause case swapping*: changing SQL clause with random case swap such as replacing "SELECT" with "sELeCt".
(10) *Swapping case*: case swapping all letters.
(11) *Double URL encode*: encoding payload twice with URL encoding.
(12) *Nesting stripped*: Embedding some clauses with themselves, e.g., replacing "SELECT" with "SELSELECTECT".
(13) *HTML encode*: encoding payload with HTML encoding.
(14) *Clause concatenation*: SQL clauses are rewritten with EXEC or CONCAT commands such as replacing "SELECT" with "CONCAT('SE', 'LECT')".
(15) *Swapping clause to alternatives*: some clauses are converted to their alternatives such as replacing "@@version" with "version()".
(16) *Overlong UTF-8*: coding a random character from the payload with overlong UTF-8, e.g., replacing "I" with "xC1A9" [25].

3.5 Reward

The reward function determines the agent's motivations. Finding the appropriate reward function is an ongoing research area. Since we assume that the WAF is a black box for the attacker (the agent), we face a sparse reward space. We define $R_t = 1$ for the HTTP status 200 (request is allowed) and $R_t = 0$ for HTTP status 403 (request is forbidden). Likewise, R_t = -1 as a penalty when receiving HTTP statuses 413 (payload too large) and 414 (URI too long).

To prevent the agent to select an action, which no longer makes a change causing a reward, we set R_t = - 0.02. We also expect to increase the exploration rate through this penalty. The WAF-Brain returns a probability value as the response to malicious payloads, called the model's confidence rate. To keep the experiment as a black-box approach, we set malicious threshold “0.5” and determine $R_t = 1$ for probabilities less or equal than the malicious threshold, and $R_t = 0$ for the greater probabilities. To persuade the evading task with the fewest possible actions, we consider an update on reward function over mutation periods as $R = R_t - (\rho \times time\_step)$ where R_t is the reward at step t inside one episode, and ρ is the constant penalty, set to 0.01 multiply the number of time-steps in each episode. Since the environment is multi-variable, it is doubtful whether the same action will earn the same reward. To compensate for this defect and the sparse reward problem, we use some enhancement techniques in both agents discussed in the following section.

3.6 Agent

We consider three agents based on three common RL algorithms for this work.

First, Q-learning as a value-based, model-free, off-policy RL algorithm is used to learn a policy that indicates what action should be taken in a given state. As stated in [26], “in Q-learning, the learned action-value function, Q, directly approximates Q*, the optimal action-value function, independent of the policy being followed”. Our Q-network agent uses deep neural networks, as feed-forward networks, with hidden layers of perceptrons to adjust the weights to extract the features of the inputs. With deep neural networks, computation to approximate the action-value function is efficient [27]. To overcome overestimating action-value, we use the double Q-learning technique that addresses this issue by utilizing two different function approximators with the same structure for action selection and action evaluation [28]. Finally, we employ prioritized experience replay. This technique is being used to answer which experiences should the agent replay from the buffer to learn efficiently [29].

The second agent is based on Advantage Actor-Critic (A2C). It is a policy gradient, model-free, and on-policy method for RL. The actor-critic method combines value-based and policy gradient methods [30]. In our agent, the critic model updates the state-value function parameters, and the actor model updates the parameters of policy in the orientation advised by the critic model. A2C is a deterministic, synchronous version of the A3C algorithm. The critic and actor networks map each state to a corresponding Q-Value and action, respectively. Advantage function computes error of agent prediction and determines if a state is better or worse than expected. A2C executes online learning, so using the current policy explores transitions and immediately updates it. The solution kept in A2C is independent of an experience replay memory like DQN and instead, uses multiple parallel actors and learners.

The third agent is based on proximal policy optimization (PPO), which is a technique for optimizing policies used in actor-critic methods. It uses several epochs of stochastic gradient ascent to perform each policy update such that minimizes the cost function while ensuring that the deviation from the previous policy is relatively small. The policy structure is known as the actor, because it is used to select actions, and the estimated value function is known as the critic because it criticizes the actions made by the actor [26]. PPO balances between ease of implementation, sample complexity, and ease of tuning. PPO, by preserving a nearly small deviation from the preceding policy, tries to compute an update in each step leads to minimizing the cost function [31]. In this work, the PPO agent also uses deep neural layers for actor and critic networks. We deploy PPO in parallel environment mode with different seeds.

Interacting with the environment, RL agents learn by trial and error. They update their behaviors using the feedback from the environment. Sufficient environment exploration helps them to better learn the policy. In many cases, common strategies like randomly choosing actions with some probability (ϵ-greedy) will eventually lead the agent to some rewards and increase the exploration rate [32]. But, in an environment with sparse rewards, like our environment, since the agent might spend most of its time without receiving any reward, there will be no appropriate exploration rate. To overcome this problem, we use Random Network Distillation (RND) [33]. In this technique, we add two neural networks with similar architecture to the agents, both get the states as input and return a vector. A randomly initialized target network remains fixed throughout training and a prediction network tries to predict the output of the target network. The prediction error of the second network considers as an approximation of how new this state is. The higher error rate suggests that this state is novel. This error decreases for the states frequently visited by the agent during training. This error is given to the agent as an intrinsic reward.

4 Experiments

4.1 Implementation

The RL environment in this work has been implemented by simulating a server protected with WAF and an adversary using the OpenAI gym toolkit for developing and comparing RL algorithms. This environment can be used to deploy evasion tasks for all types of WAFs. In this work, we consider ModSecurity-CRS and Naxsi, which are two of the best-known open-source and rule-based solutions, as a module on the Nginx web server. Naxsi, stands for Nginx Anti XSS & SQL injection, is a third-party module for Nginx web server and reverse proxy. Its developers see its strength as the simplicity and readability of the rules and its high resistance to circumvention. We configure Naxsi in the default mode, which prescribes each suspicious request to SQL injection with a score greater than or equal to 8 must be blocked. We also used whitelists to avoid false positives [34]. As an example of open-source ML-based WAF, we use WAF-Brain [35], which has been also used in [14] and [16]. This WAF applies the RNN algorithm to classify SQL injection payloads.

All the payloads used in our framework are classified as malicious with all the three WAFs. Our framework is implemented in Python 3.8.8 and PyTorch 1.8.0, which is an open-source ML library based on the Torch library. To implement action-value function Q, target action-value function Q* in Q-learning, and actor-critic in A2C and PPO we utilize a deep neural network.

In the context of the WAF evasion task, after passing the malicious payload to the embedding model, the actor-network takes the feature extracted as input, then predicts the next action the agent will take. In our work, the input dimension is the pair (number of tokens, lengths of embedding vector) and the output dimension equals a vector of size 16. DDQN, A2C, and PPO that we have designed have a common structure in model layers. There are two hidden layers each of which is followed by a Tanh activation layer and one output layer at the end. The hyperbolic tangent (Tanh) is an activation function that outputs values in the range $[-1..1]$. Although ReLU shows significantly rapid convergence over traditional activation functions and robustness in terms of gradient vanishing problems [36], Tanh has better results in our experiments. We utilize Adam as an optimizer for all the three agents. The optimizers are algorithms for modifying the neural network attributes such as learning rate and weights to reduce losses. Adaptive Moment Estimation (Adam) is an optimizer, which utilizes both momentum and scaling, and is efficient when the problem involves a lot of data or parameters [37]. PPO and A2C agents have also one SoftMax layer after the output layer in the actor-network.

4.2 Training with DDQN

Training in RL algorithms is the process of using data to estimate the optimal policy. We apply the double deep Q-learning with a prioritized experience replay algorithm against ModSecurity-CRS, Naxsi, and WAF-brain. The agent makes transitions between states by performing actions. To explore the environment, an ϵ-greedy strategy has been employed. To allow the agent to aim for high rewards over a long run, the discount factor γ is set to 0.99. The step size at each iteration or learning rate is set to 0.001 for ModSecurity and WAF-Brain, and 0.0005 for Naxsi. Also, we use the PyTorch learning scheduler to converge faster and avoid plateaus. To update the target network, we use the "soft update" method. This method, instead of updating the network at once after every few steps, updates the network slightly in each step with τ=0.001. Every 20 steps, the prioritized experience replay buffer has been updated to ensure that transitions with higher rewards have a higher priority to replay. We run training for 5000 episodes with 32 maximum time-steps in all the three agents, although some episodes end early. More details about major parameters and their values for our training algorithms are shown in Table 2.

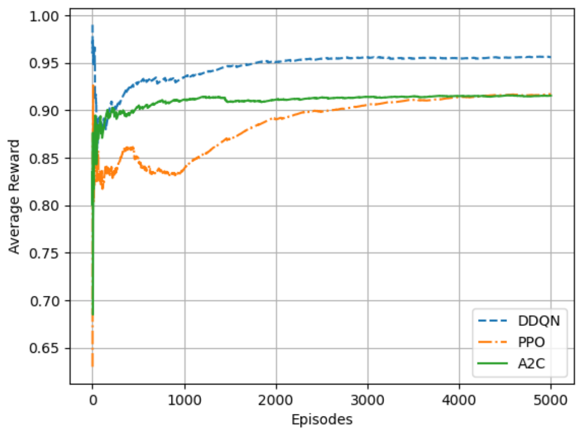

Figure 3. The average reward in training DDQN, A2C, and PPO over ModSecurity-CRS (Paranoia level 1)

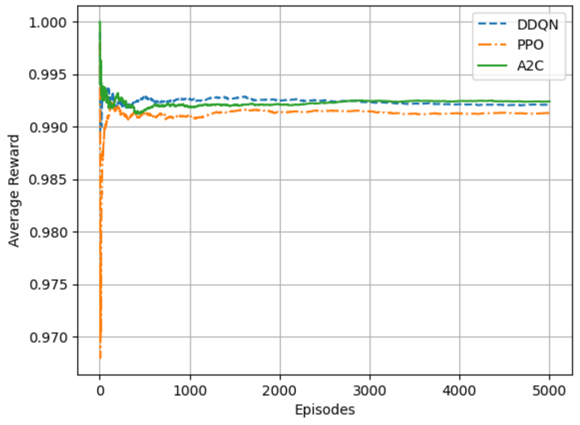

Figure 4. The average reward in training DDQN, A2C, and PPO over WAF-Brain

Table 2. List of parameters and their values in the training algorithms

Agent	Parameter	ModSecurity	WAF-Brain	Naxsi	Description
Common	Max-Timesteps	32	32	32	All three agents are allowed to perform up to "Max-Timesteps" steps mutations in each episode
Common	Learning-rate	0.001	0.001	0.005	The step size at each iteration or learning rate
Common	Discount factor (gamma)	0.99	0.99	0.99	Discount factor γ determines importance of future rewards
DDQN	Hidden layer dimension	64	64	16	Number of hidden neurons
DDQN	Mini-batch-size	16	16	4	The number of training case for each network update
DDQN	Buffer size	1e6	1e6	1e6	The capacity of experience replay buffer
DDQN	Initial epsilon	1	1	1	Start rate of exploration
DDQN	Final epsilon	0.01	0.01	0.01	Minimum rate of exploration
DDQN	Epsilon decay	0.999	0.999	0.999	Decay factor for exploration rate
DDQN	update frequency for prioritized experience replay	20	20	20	how often to update the priorities
PPO	Hidden layer dimension	64	128	64	Number of hidden neurons
PPO	Mini-batch-size	16	16	32	The number of training case for each network update
PPO	Horizon for update PPO (episodes)	8	8	8	when episode reaches to this number, PPO network starts to training
PPO	PPO epochs	20	20	20	How many epochs per update
PPO	Horizon for update RND (steps)	128	128	128	when steps reach to this number, RND network starts to training
PPO	KL-divergence range	0.0008	0.0008	0.0008	Quantifies how much one probability distribution differs from another probability distribution
PPO	Entropy coefficient	0.001	0.001	0.001	How much randomness of action you will get
PPO	Value function coefficient	1	1	1	Coefficient for value loss
PPO	Lambda	0.95	0.95	0.95	Generalized Advantage Estimation (GAE) parameter
PPO	Number of parallel environments	4	4	4	Number of parallel environments with different seeds
A2C	Hidden layer dimension	16	128	128	Number of hidden neurons
A2C	Mini-batch-size	8	4	32	The number of training case for each network update
A2C	Entropy coefficient	0.001	0.001	0.001	How much randomness of action you will get
A2C	Value function coefficient	0.5	0.5	0.5	Coefficient for value loss
A2C	Number of workers	4	4	4	Number of workers

4.3 Training with A2C

We also apply the A2C algorithm against all the three WAFs. We set the entropy coefficient to be 0.001 and the value function coefficient to be 0.5. High entropy occurs when all values of one output are equal and low entropy occurs when one value of output has a much

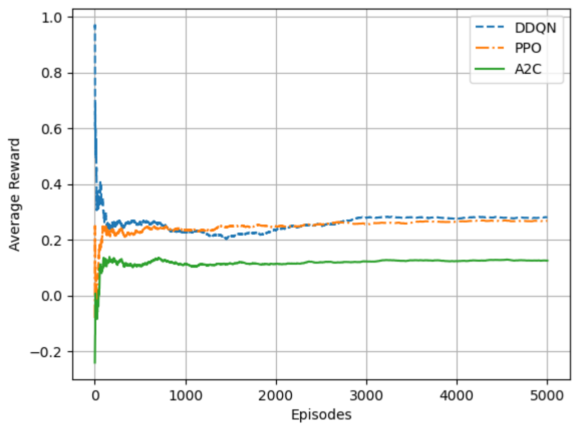

Figure 5. The average reward in training DDQN, A2C, and PPO over Naxsi

higher value than the others. The entropy coefficient is the extent to which we maximize entropy from the agent's output. We try to maintain a little entropy, so the agent can explore further.

4.4 Training with PPO

Currently, PPO is the default RL algorithm in OpenAI because of its ease of use and good performance. Therefore, we decided to try our experiments with this algorithm, as well. As stated in Section 3.6, we designed a PPO agent with the RND technique for improving the exploration rate. We set the discount factor γ to be 0.99, λ to be 0.95, learning rate to be 0.001, clip range to be 1, entropy coefficient to be 0.001, and KL-divergence range to be 0.0008. The parameters in the RND prediction network and PPO network have been updated for every 32 steps and 8 episodes, respectively. We also use dropout and L2 regularization to overcome overfitting in the PPO network. Figure 3, Figure 4, and Figure 5 show the agent's improvement after training based on an average reward that the DDQN, A2C, and PPO agents won against ModSecurity-CRS at default level (paranoia-1), Naxsi, and WAF-Brain, respectively.

4.5 Tuning Hyperparameters

Due to finding appropriate hyperparameters, we performed multiple experiments with all the three algorithms. The experiments were performed on ModSecurity-CRS for 10000 episodes. Some of the experiments are explained in this section.

For DDQN we tried multiple mini-batch sizes such as 8, 16, 32 and found 16 as a suitable value (Figure 6). Also, we examined the training reward for different entropy coefficient values such as 0, 0.001, and 0.0001 in PPO, and found 0.001 as a proper value (Figure 7). In some experiments, the average reward for the DDQN agent decreases after a period of training, which might be a consequence of using the prioritized experience replay technique. This is because the model in this technique is biased towards a subset of the state space with a higher temporal difference error. Since a small set of states have very large errors, the model performs poorly for the remainder of the state space, especially in long training. Figure 8 shows the mean magnitude of the policy loss function in training DDQN over WAF-Brain, which demonstrates policy change. The magnitude should be reduced during a successful training process.

5 Results

Testing in RL is the process of evaluating the policy obtained by training. To evaluate in supervised learning, we may measure accuracy on a labeled dataset, and consider a classifier as good if its accuracy is higher. However, the evaluation in RL is more challenging. The state depends on the selected action by the agent. Therefore, if the agent, with respect to the learned policy, chooses a different action in comparison with the trained data, it encounters trajectory tuples, which have not been seen before [38].

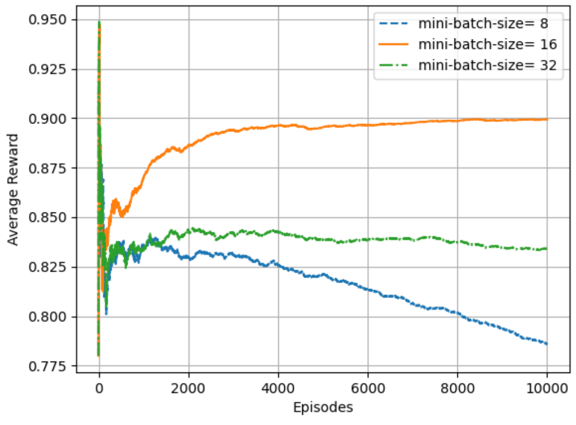

Figure 6. Comparing average reward in training DDQN agent with different mini-batch sizes over ModSecurity-CRS

Table 3. Success evasion rate for evaluation policy (1024 samples)

WAF \ Agents	Random	DDQN	PPO	A2C
ModSecurity-CRS-Paranoia Level 1	89.84%	94.23%	76.36%	**97.46%**
NAXSI	43.35%	**62.5%**	42%	49.31%
WAF-Brain	99%	97.85%	95.31%	**100%**

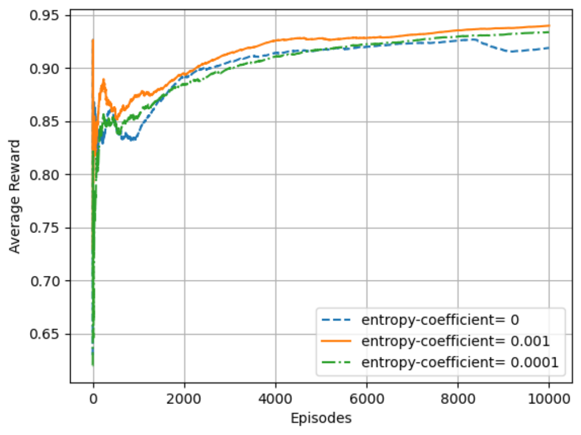

Figure 7. Comparing average reward in training PPO agent with different entropy coefficient values over ModSecurity-CRS

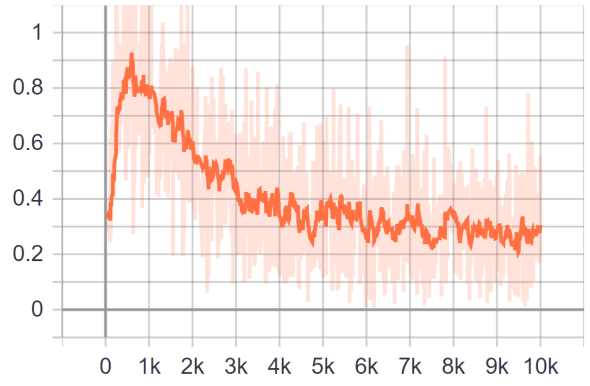

Figure 8. Mean magnitude of policy loss function in training DDQN agent over ModSecurity-CRS

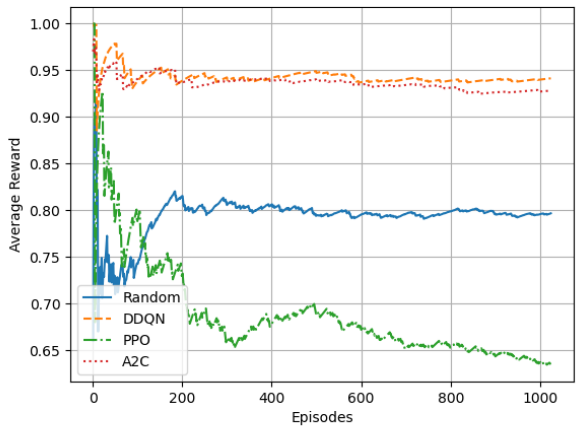

Figure 9. Comparing average reward in evaluating learned policies with DDQN, A2C, and PPO agents against random agent in ModSecurity-CRS

The average obtained reward usually measures the quality of a policy if the agent follows the policy to select actions. In our environments, evaluation can be done by running the policy. To evaluate the policy,

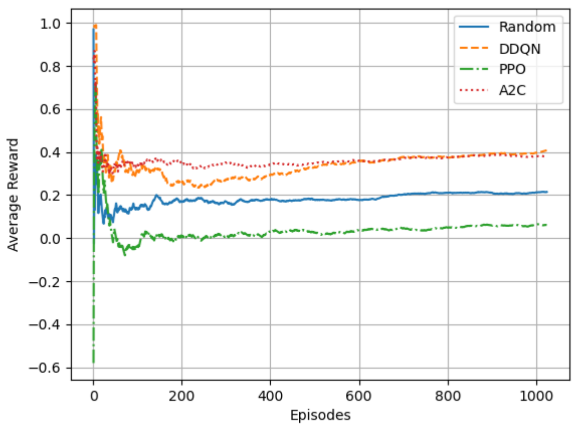

Figure 10. Comparing average reward in evaluating learned policies with DDQN, A2C, and PPO agents against random agent in Naxsi

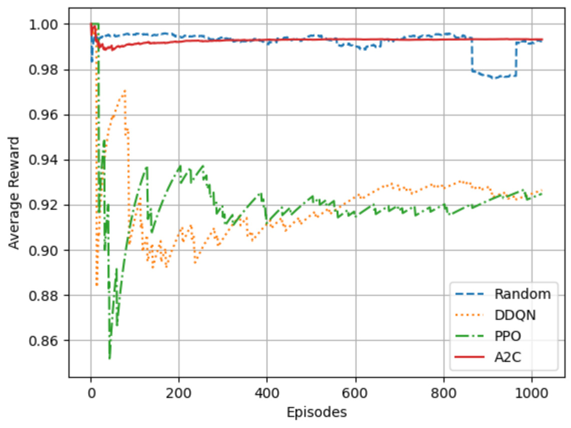

Figure 11. Comparing average reward in evaluating learned policies with PPO, A2C, and DDQN agents against random agent in WAF-Brain

a random agent with an exploration rate = 1 must be defined so that its behavior can be used as a baseline. The random agent uses the same initial seed, which RL agents use in the training and evaluation processes. It should be noted that the initial seed might be varying for each environment depending on which seed leads to a better result in the training phase. In the evaluation process, the agents examine their policy against over the 2^{10} unseen SQL injection payloads sampled randomly from the dataset in [14]. The agents are allowed to perform up to 32 mutation steps before declaring failure.

In general, results show that all the three agents have been successful against the WAFs. Figure 9, Figure 10, and Figure 11 show the average reward that agents win against ModSecurity-CRS, Naxsi, and WAF-Brain. To evaluate, we set the epsilon value to “0.05” in deep Q-learning. Obtained rewards in training and evaluating for our agents, as shown in Figure 5 and Figure 10, state that resistance of Naxsi

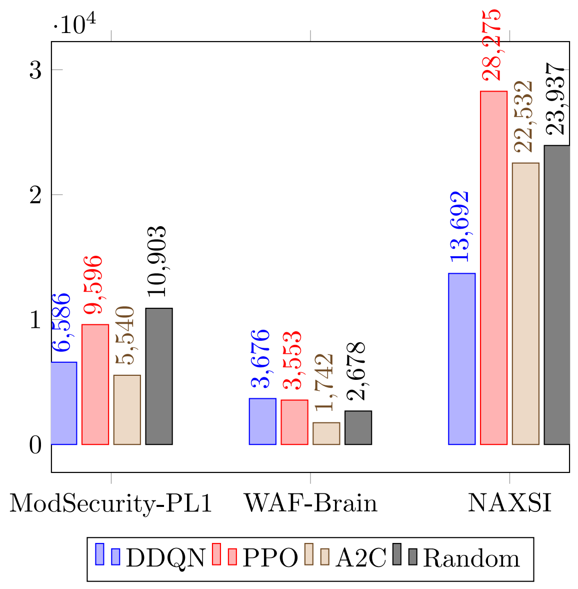

Figure 12. Comparing number of steps for every agent in evaluation against ModSecurity-CRS-PL1, NAXSI, and WAF-Brain

Table 4. An example of a mutation sequence

Malicious Payload	UPDATE ‘tab‘ SET ‘col2‘ = 7 WHERE ‘col1‘ ¡ ”’) AND 7331=(SELECT COUNT(*) FROM sysusers AS sys1,sysusers AS sys2, sysusers AS sys3,sysusers AS sys4, sysusers AS sys5,sysusers AS sys6, sysusers AS sys7) AND (”fkQy”=”fkQy”’)

Target WAF	**Agent**	**Action Sequence**
ModSecurity-CRS	Random	2,5,9,6,9,3,12,1,1,9 ,10,14,14,10,14,2,4,14,4,5,6,7,1
ModSecurity-CRS	DDQN	11
ModSecurity-CRS	PPO	11
ModSecurity-CRS	A2C	3,4,8,14,7,9,12,3,16,5,9,11
NAXSI	Random	5,15,16,14,1,8,12,14,2,14, 14,5,11
NAXSI	DDQN	11,14,11
NAXSI	PPO	11
NAXSI	A2C	failed
WAF-Brain	Random	15,8,7,12
WAF-Brain	DDQN	8,14,8,8,8,8,8
WAF-Brain	PPO	11
WAF-Brain	A2C	8,7,13,14

against crafted malicious payloads is higher than two other WAFs. We can see from Figure 11 that WAF-Brain has low resiliency over the mutated payloads and quickly loses confidence. Also, because of the enrichment action set, the random agent performs well against WAFs and defeats two of the three agents in the WAF-Brain evasion task. As it is seen in Table 3, the A2C agent has the most evasion rate against ModSecurity with 97.46% and WAF-Brain with 100%. Also, the DDQN agent has the most evasion rate against Naxsi.

In our approach, the success evasion rate for the learned policy in PPO agent is 76.36% and 95.31% for ModSecurity and WAF-Brain, respectively. In [16], for comparison, the success evasion rate is 8.7% and 20% for ModSecurity and WAF-Brain, respectively. In [14], WAF-Brain is also used as an environment. However, comparable numerical results to compare with our approach are not provided. Other related works are not comparable in terms of results due to differences in the environment.

Figure 12 indicates the total number of steps that agents take to pass the evaluation process. In the ModSecurity-CRS environment, the A2C agent takes about one-half steps than the random agent. But in general, although we expect that the random agent has taken many more steps to solve the problem, this is not the case. Also, we can see the total number of steps for Naxsi WAF circumvention compared to the other two WAFs has significantly increased. Table 4 shows the sequence of actions for a sample payload. Each agent has been able to bypass the WAF after performing this actions sequence. We have 16 actions listed in Section 3.4.

6 Conclusion and Future Works

In this work, we showed that RL can be a solution to the WAF evasion problem. To solve the problem, we designed a WAF evasion environment and deployed a deep double Q-learning algorithm with prioritized experience replay technique, advantage actor-critic algorithm, and proximal policy optimization algorithm with random network distillation technique as evader agents. Our testbeds for training and evaluating were ModSecurity-CRS, Naxsi, and WAF-Brain. We evaluated the policy that agents learned against a set of unseen payloads and compare it with a random agent as a baseline. The result provides proof-of-concept support to the hypothesis that RL techniques may not yet be as reliable as brute-force techniques, but they have the potential to be considered as a solution used in the future to tackle the WAF evasion problem.

Future directions to extend this work includes deploying this framework against more ML models proposed in the academy and other open-source web application firewalls with different techniques such as hybrid ones. It is also possible to get better results by different adjustments of agents and searching the hyperparameter space. In this work, we used a deep neural network with linear layers. This layer can be replaced with other neural networks such as CNN or LSTM in future works. Finally, we plan to cover other types of web application vulnerabilities such as cross-site-scripting and try to improve current mutation functions.

References

[1] Web application firewall. https://owasp.org/www-community/Web_Application_Firewall. Accessed: 2021-12-24.

[2] Ali Moradi Vartouni, Mohammad Teshnehlab, and Saeed Sedighian Kashi. Leveraging deep neural networks for anomaly-based web application firewall. *IET Information Security*, 13(4):352–361, 2019.

[3] Mojtaba Hemmati and Mohammad Ali Hadavi. Using deep reinforcement learning to evade web application firewalls. In *2021 18th International ISC Conference on Information Security and Cryptology (ISCISC)*, pages 35–41. IEEE, 2021.

[4] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J Doug Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58, 2011.

[5] Guillermo Caminero, Manuel Lopez-Martin, and Belen Carro. Adversarial environment reinforcement learning algorithm for intrusion detection. *Computer Networks*, 159:96–109, 2019.

[6] Bhagyashree Deokar and Ambarish Hazarnis. Intrusion detection system using log files and reinforcement learning. *International Journal of Computer Applications*, 45(19):28–35, 2012.

[7] Di Wu, Binxing Fang, Junnan Wang, Qixu Liu, and Xiang Cui. Evading machine learning botnet detection models via deep reinforcement learning. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[8] Hyrum S Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. Learning to evade static pe machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917*, 2018.

[9] Zhiyang Fang, Junfeng Wang, Boya Li, Siqi Wu, Yingjie Zhou, and Haiying Huang. Evading anti-malware engines with deep reinforcement learning. *IEEE Access*, 7:48867–48879, 2019.

[10] Konstantin Pozdniakov, Eduardo Alonso, Vladimir Stankovic, Kimberly Tam, and Kevin Jones. Smart security audit: reinforcement learning with a deep neural network approximator. In *2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, pages 1–8. IEEE, 2020.

[11] Fabio Massimo Zennaro and Laszlo Erdodi. Modeling penetration testing with reinforcement learning using capture-the-flag challenges: trade-offs between model-free learning and a priori knowledge. *arXiv preprint arXiv:2005.12632*, 2020.

[12] Mohamed C Ghanem and Thomas M Chen. Reinforcement learning for efficient network penetration testing. *Information*, 11(1):6, 2019.

[13] László Erdődi, Åvald Åslaugson Sommervoll, and Fabio Massimo Zennaro. Simulating sql injection vulnerability exploitation using q-learning reinforcement learning agents. *Journal of Information Security and Applications*, 61:102903, 2021.

[14] Luca Demetrio, Andrea Valenza, Gabriele Costa, and Giovanni Lagorio. Waf-a-mole: evading web application firewalls through adversarial machine learning. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 1745–1752, 2020.

[15] Dennis Appelt, Cu D Nguyen, Annibale Panichella, and Lionel C Briand. A machine-learning-driven evolutionary approach for testing web application firewalls. *IEEE Transactions on Reliability*, 67(3):733–757, 2018.

[16] H. Hu X. Wang. Evading web application firewalls with reinforcement learning. https://openreview.net/pdf?id=m5AntlhJ7Z5. Accessed: 2021-12-24.

[17] Dennis Appelt, Cu D Nguyen, and Lionel Briand. Behind an application firewall, are we safe from sql injection attacks? In *2015 IEEE 8th international conference on software testing, verification and validation (ICST)*, pages 1–10. IEEE, 2015.

[18] Gym-waf. https://github.com/sanebow/gym-waf. Accessed: 2021-12-24.

[19] Libinjection. https://github.com/client9/libinjection. Accessed: 2021-12-24.

[20] Modsecurity-nginx. https://github.com/SpiderLabs/ModSecurity-nginx. Accessed: 2021-12-24.

[21] Coreruleset. https://github.com/coreruleset/coreruleset. Accessed: 2021-12-24.

[22] A03:2021 – injection. https://owasp.org/Top10/A03_2021-Injection/. Accessed: 2021-12-24.

[23] P Bojanowski. Grave e joulin a mikolov t. *Enriching word vectors with subword information TACL*, 5:135–146, 2017.

[24] Naxsi. https://github.com/nbs-system/naxsi. Accessed: 2021-12-24.

[25] Kevin Boone. Utf-8 and the problem of over-long characters. https://kevinboone.me/overlong.html?i=1. Accessed: 2021-12-24.

[26] Richard S Sutton and Andrew G Barto. Rein-

formcent learning: An introduction, 1998.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[28] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[29] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[30] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[31] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[32] Or Rivlin. Reinforcement learning with exploration by random network distillation. https://towardsdatascience.com/reinforcement-learning-with-exploration-by-random-network-distillation-a3e412004402. Accessed: 2021-12-24.

[33] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[34] nxutil. https://github.com/prajal/nxutil. Accessed: 2021-12-24.

[35] Waf-brain. https://github.com/BBVA/waf-brain. Accessed: 2021-12-24.

[36] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[37] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] Lihong Li. A perspective on off-policy evaluation in reinforcement learning. *Frontiers of Computer Science*, 13(5):911–912, 2019.

Mojtaba Hemmati received his B.Sc. degree in Software Engineering from Islamic Azad University, Sari Branch, Iran. He then received his M.Sc. degree in Secure Computing from Malek Ashtar University of Technology, Tehran, Iran in 2021. His research interests include software security, application security, machine learning in security and adversarial machine learning.

Mohammad Ali Hadavi received his Ph.D. degree in Computer Engineering from Sharif University of Technology, Tehran, Iran, in 2015. He received his M.Sc. and B.Sc. degrees in Software Engineering from Amirkabir University of Technology, Tehran, Iran, in 2004, and from Ferdowsi University of Mashhad, Iran, in 2002, respectively. Now, he is an assistant professor at Malek Ashtar University of Technology. Focusing on information security, he has published more than 30 papers in national and international journals and conference proceedings. His research interests include software security, database security, and security aspects of data outsourcing.