


Article

Deep Reinforcement Learning for Intelligent Penetration Testing Path Design

Junkai Yi ¹  and Xiaoyan Liu ^{2,*}¹ School of Automation, Key Laboratory of Modern Measurement and Control, Technology Ministry of Education, Beijing Information Science and Technology University, Beijing 100192, China; yijk@bistu.edu.cn² School of Automation, Beijing Information Science and Technology University, Beijing 100192, China

* Correspondence: 2021020435@bistu.edu.cn

Abstract: Penetration testing is an important method to evaluate the security degree of a network system. The importance of penetration testing attack path planning lies in its ability to simulate attacker behavior, identify vulnerabilities, reduce potential losses, and continuously improve security strategies. By systematically simulating various attack scenarios, it enables proactive risk assessment and the development of robust security measures. To address the problems of inaccurate path prediction and difficult convergence in the training process of attack path planning, an algorithm which combines attack graph tools (i.e., MulVAL, multi-stage vulnerability analysis language) and the double deep Q network is proposed. This algorithm first constructs an attack tree, searches paths in the attack graph, and then builds a transfer matrix based on depth-first search to obtain all reachable paths in the target system. Finally, the optimal path for target system attack path planning is obtained by using the deep double Q network (DDQN) algorithm. The MulVAL double deep Q network (MDDQN) algorithm is tested in different scale penetration testing environments. The experimental results show that, compared with the traditional deep Q network (DQN) algorithm, the MDDQN algorithm is able to reach convergence faster and more stably and improve the efficiency of attack path planning.

Keywords: deep reinforcement learning; penetration testing; attack graph; improved DQN algorithm; attack path planning



Citation: Yi, J.; Liu, X. Deep Reinforcement Learning for Intelligent Penetration Testing Path Design. *Appl. Sci.* **2023**, *13*, 9467. <https://doi.org/10.3390/app13169467>

Academic Editors: Yutaka Ishibashi and Aleksander Mendyk

Received: 9 July 2023

Revised: 10 August 2023

Accepted: 19 August 2023

Published: 21 August 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the rapid development of the Internet and information technology, traditional security defense methods have been unable to cope with increasingly complex network security threats. Penetration testing [1] is an important method to assess the security degree of network systems. It simulates invasive behavior from the perspective of the attacker, finds the attack path hidden in the system, and then evaluates the security performance of the system. However, manual penetration testing requires strong professional skills. In recent years, automated penetration testing has become one of the “hotspots” in the network security field. As the key step in automated penetration testing, attack path planning is of great significance. Penetration testing path design refers to the process of strategically mapping out potential routes an attacker might take to exploit vulnerabilities within a target system. This involves analyzing the system’s architecture, identifying weak points, and determining the sequence of steps an attacker could follow to compromise its security. The goal is to simulate real-world attack scenarios, assess potential risks, and enhance the system’s defenses by proactively addressing these vulnerabilities.

Almazrouei et al. [2] used an attack graph model to find attack paths. This model is widely used in assessing network vulnerabilities. However, because of the state explosion problem, it is more suitable for small network penetration testing scenarios. To conduct penetration testing in large-scale network environments, a penetration scenario is transformed

into the planning domain definition language (PDDL), and classical planning algorithms are used to discover the attack paths. The penetration testing network environment is modeled by description files of the network configuration and vulnerability information [3–5]. The basic idea of this kind of method is to transform the penetration testing problem into PDDL, and then to obtain the attack path by using an artificial intelligence planning algorithm.

To find the best attack path in the penetration testing of uncertain state space, the probability is incorporated into the classical planning algorithm, and a partially observable Markov decision process (POMDP) is used to model the attack path planning problem. This is because the action uncertainty should be considered in the attack path planning problem [6,7]. The POMDP model has achieved success in academic applications and solved some problems caused by the uncertainty state space of penetration testing, but there are still some difficulties in achieving accurate solutions. There are significant limitations to the scale of the problem that can be solved using POMDP [8], which makes it difficult to apply to real penetration testing situations.

The Markov decision process (MDP) [9–11] treats penetration testing as a state transition, reflecting the uncertainty of penetration testing. Some reinforcement learning algorithms are modeled by an MDP and are trained by interacting with the environment. Then, the policy for attack path planning is generated and the goal of maximizing long-term reward can be achieved. At the same time, the MDP is more generalizable than the POMDP model because of its lower computational complexity. Gangupantulu et al. [12] presented methods for constructing attack graphs using notions from IPB on a cyber terrain, introducing the cyber terrain by modifying the state transition probabilities and reward function. The methodology proposed maintains an automated, scale-oriented approach to constructing MDPs, while introducing notions of a cyber terrain that help ground reinforcement learning agent behavior to reality.

The traditional deep Q network (DQN) algorithm is commonly used in reinforcement learning. It was the first mature algorithm to combine deep learning and reinforcement learning [13]. The deep Q network algorithm has demonstrated better performance than the previous algorithm in many experiments [14,15]. The value of Q is easily overestimated, which leads to suboptimal policy updates and divergent behavior. Wang et al. [16] improved the deep Q network algorithm by adopting a dueling network mechanism and using the classical Q learning algorithm to learn the conservative Q function. In this way, the training efficiency was improved. Tran et al. [17] proposed the HA-DRL algorithm, which uses a hierarchical deep reinforcement learning architecture. It uses an algebraic action decomposition strategy to solve the problem of the large discrete action space of penetration testing, finding a stable optimal attack strategy more quickly.

In recent years, with the continuous development of deep reinforcement learning [18], more and more researchers have applied deep reinforcement learning to penetration testing and achieved good results. Yang et al. [19] introduced a coverage-based masking mechanism that reduced attention on previously selected actions to help the agent adapt to future exploration. Tran et al. [20] employed an algebraic action decomposition strategy. The method proposed was able to find the optimal attack policy in scenarios with large action spaces faster and more stably than a conventional deep Q learning agent. Cody et al. [21] built on the previous crown jewels (CJ) identification that focused on the target goal of computing optimal paths that adversaries may traverse toward compromising CJs or hosts within their proximity. Ghanem et al. [22] proposed an approach called the intelligent automated penetration testing framework (IAPTF), utilizing model-based RL to automate sequential decision making. The IAPTF with hierarchical network modeling outperformed previous approaches as well as human performance in terms of time, the number of tested vectors, and accuracy, with the advantages increasing with network size. The Summary of typical algorithms is shown in Table 1.

In summary, the main contributions of this paper are as follows:

- An MDDQN algorithm for improving DQN is proposed. The MDDQN algorithm integrates attack graphs with the double deep Q network (DDQN) algorithm to address the problem of intelligent penetration testing path design.

- The MDDQN algorithm partly solves the sparse reward problem that prevails in reinforcement learning algorithms. It provides more positive rewards to the DDQN algorithm by using prior knowledge provided by the attack graph.

Table 1. Summary of typical algorithms.

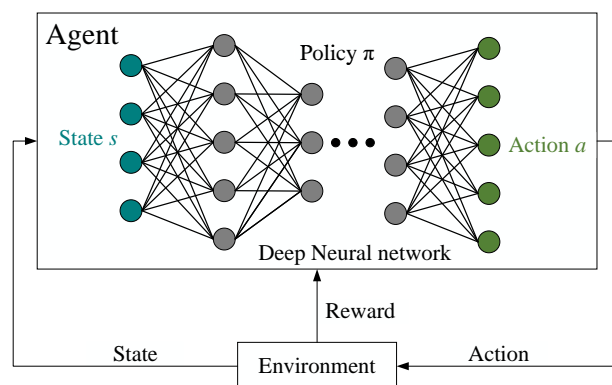
Authors	Model	Method Used
Zhang et al. [6]	POMDP	Improved deep recurrent Q network
Yang et al. [8]	POMDP	AAM-extended POMDP approach
Cody [10]	POMDP	A layered reference model with RL and attack graph
Wang et al. [16]	MDP	DUSC-DQN
Tran et al. [20]	MDP	Deep cascaded reinforcement learning agents
Hu et al. [23]	MDP	Deep Q network with attack graph

The rest of this paper is organized as follows: Section 2 introduces the research background of deep reinforcement learning and improved deep Q network algorithms; Section 3 proposes an attack path planning method that combines the attack graph with the double deep Q network algorithm; Section 4 discusses the experimental results and analysis; Section 5 presents the conclusions of the experiment and discusses future work.

2. Related Work

2.1. Deep Reinforcement Learning

As an artificial intelligence method, deep reinforcement learning combines deep learning [24,25] and reinforcement learning [26,27]. The agent has strong perception ability in deep learning and strong decision-making in reinforcement learning. Deep reinforcement learning complements the advantages of both and the basic principle of deep reinforcement learning is shown in Figure 1.

**Figure 1.** Basic principle of deep reinforcement learning.

Deep reinforcement learning leverages deep neural networks as function approximators to represent the agent's policy or value function. These neural networks can learn complex patterns and representations, allowing agents to handle high-dimensional and continuous state spaces. Compared with traditional reinforcement learning, deep reinforcement learning greatly improves the efficiency of agent training.

This advancement has enabled deep reinforcement learning to excel in domains such as mechanical control, network security, autonomous driving, healthcare, finance, and others [28]. It has demonstrated remarkable success in training robots to perform intricate tasks, optimizing financial portfolios, and addressing other complex challenges across different fields. Deep reinforcement learning is still developing at high speed, with researchers continuously proposing algorithms with better training effects and applying them to an increasingly wide range of fields [29].

2.2. Improved Deep Q Network Algorithm

DQN [30–32] is the first mature algorithm that combines deep learning and reinforcement learning, and is an extension of the Q-learning algorithm. The Q-learning algorithm initializes the Q-table with a limited number of states and actions and continuously updates them, but it is prone to the problem of state explosion.

The DQN algorithm adopts the experience replay method and updated method with network delay to solve the problems of learning instability and excessive sample correlation in the Q-learning algorithm.

The experience replay method stores the past training data. When the parameters are updated, a part of the data is exacted from the memory in order to solve the problem of the high correlation between data.

The updated method with network delay creates the target Q network. During the update process, only the parameter of the current Q network is updated, and the θ parameter of the target Q network remains unchanged. After a certain number of iterations, the updated current Q network is copied to the target Q network.

The target value is relatively fixed for a period of time when the target Q network does not change. Therefore, the introduction of the target Q network increases the stability of the DQN algorithm learning. In each episode of DQN algorithm training, the network parameters will be updated through gradient descent backpropagation to minimize the loss function.

The DQN algorithm uses the Q network approximation function to solve the continuous state space problem. However, it still has the problem of overestimating the Q value. To improve the calculation method of the target and deal with this problem, the DDQN is proposed. The DDQN algorithm is an improvement of the DQN algorithm. The structure of the DDQN algorithm is similar to the DQN algorithm, but the target function is improved, as shown in Equations (1) and (2).

$$Y_t^{DQN} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t^-) \quad (1)$$

$$Y_t^{DDQN} = R_{t+1} + \gamma Q(S_{t+1}, \arg \max_a Q(S_{t+1}, a; \theta_t), \theta_t^-) \quad (2)$$

where R_{t+1} denotes the reward for the next state, γ represents the discount factor, θ_t^- denotes the neural network parameters in the target Q network, and θ_t represents the parameters of the current Q network.

The optimal action selection of the objective function in the DDQN algorithm is based on the Q network parameters that are currently being updated instead of selecting the maximum value from the target Q network. Therefore, this algorithm solves the overestimation problem of the DQN algorithm to a certain extent, making the Q value closer to the true value.

In recent years, more and more improved DQN algorithms have been proposed. Haarnoja et al. [33] proposed the soft actor-critic (SAC) algorithm. It handles continuous action spaces and incorporates an entropy regularization term to encourage exploration. Hessel et al. [34] proposed the Rainbow algorithm, which combines several DQN extensions into a unified framework. It incorporates improvements such as double Q learning, prioritized experience replay, dueling architectures, distributional reinforcement learning, and noisy networks.

3. Proposed Method

3.1. Penetration Testing Network Model

In order to use deep reinforcement learning in penetration testing, the network model for penetration testing needs to be modeled as an MDP. The MDP model is generally described in the form of quintuple $\langle S, A, P, R, \gamma \rangle$ and policy π .

- S represents the set of network environment states, that is, the information obtained by the agent from the environment, including all host information, such as host

identification, host privileges, vulnerability information, running services, and the asset values of hosts.

- A represents the set of actions, that is, the set of operations that the agent performs on the host, including file access, vulnerability exploitation, and privilege escalation to the host.
- P denotes the set of transition probabilities between the states and actions. $P(s' | s, a)$ denotes the probability that the state transfers to s' when the agent is in the state s and takes actions a .
- R denotes the reward function. $R(s' | s, a)$ denotes the reward for the state transfers to s' when the agent is in the state s and takes action a , and $R(s)$ represents the reward value obtained by the agent after reaching the state s .
- $\gamma \in [0,1]$ is the discount factor, which reflects how much the agent emphasizes long-term returns. The larger the discount factor, the more important the agent will be for future long-term returns.

In this process, the agent selects actions based on the policy π . As shown in Equation (3), the quality of the policy depends on the cumulative rewards obtained after long-term execution. The goal of the agent is to find an optimal strategy to maximize the cumulative rewards.

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (3)$$

By applying MDP principles, such as state representation, action space, rewards, and iterative algorithms, MDP-based penetration testing models provide a systematic framework for modeling and optimizing attacker's decisions in penetration testing.

3.2. MDDQN Algorithm

MulVAL (multi-stage vulnerability analysis language) [35] is an open source attack graph generation tool for generating the actual attack tree corresponding to a given network topology. It is designed to analyze the security of large-scale networks by identifying vulnerabilities and potential attack paths. MulVAL operates by using the Datalog language to describe the network topology and vulnerabilities. It allows for the definition of hosts, services, vulnerabilities, and the relationships among them. By specifying the network configuration and associated vulnerabilities, MulVAL can generate an attack graph that can reflect the network topology and vulnerability exploitation of the target system.

We convert the information of the target system into Datalog clauses as follows:

$$vulExists(webServer, 'CAN - 2019 - 17571', httpd) \quad (4)$$

Namely, a vulnerability with CVE ID CAN-2019-17571 was identified on the machine webserver. We convert the effect of the vulnerability into Datalog clauses such as

$$vulProperty('CAN - 2019 - 17571', remoteExploit, privilegeEscalation). \quad (5)$$

The vulnerability enables a remote attacker to execute arbitrary code with all the privileges.

In the attack graph generated by MulVAL, the root node represents the final target of the attack. Each child node represents an attack behavior or a sub-target.

The flowchart of the MDDQN algorithm is shown in Figure 2. The MDDQN algorithm first generates the MulVAL attack graph based on the known target system environment information, including the target system host information and vulnerability information. Then, it obtains all exploitable paths in the target system and builds the transfer matrix by depth-first search of the attack graph.

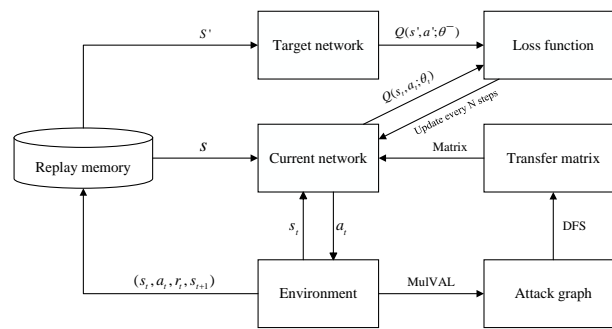


Figure 2. Flowchart of MDDQN.

The transfer matrix P_{nn} is shown in Equation (6), where, if state i to state j is not reachable, then $a_{ij} = -1$; if state i to state j is reachable, then the value of that is shown in Equation (12).

$$P_{nn} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & a_{ij} & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \quad (6)$$

When dealing with multi-subnet networks, MulVAL often generates attack graphs with higher complexity. Because the neural networks have good processing capabilities for large-scale data, the MDDQN algorithm combines neural networks with MulVAL, which can better deal with the potential state explosion problem in attack path planning. The DDQN algorithm has blind exploration behavior at the early stage of training, and the attack graph can provide a priori information for the DDQN algorithm to improve the training efficiency.

In the training process of the DDQN algorithm, it firstly initializes the parameters of the neural network, the current Q network and the target Q network parameters, and uses the ϵ -strategy to select an action. Then, it executes the action and observes the reward, the next state, and whether it is the end state. Next, it stores the (s, a, r, s') quads into the memory pool.

Agent exploration is performed with probability ϵ at a time, and the application is performed with $1 - \epsilon$. ϵ decreases throughout the training. At the beginning, the training agent focuses on exploration, while later in training it focuses on application.

$$\epsilon = \max\{\epsilon_{\min}, 1 - \frac{1 - \epsilon_{\min} * \text{step}}{\text{total step}}\} \quad (7)$$

The current Q network parameters update, randomly drawing small batches of samples from the memory pool. The Q value can be calculated according to Equation (8) and the loss function. Then, the agent updates the current Q-network parameters by using gradient descent backpropagation of the neural network.

$$y_j = \begin{cases} r_j & \text{If } s' \text{ is terminal} \\ r_j + \gamma Q(s', \max_a Q(s'; \theta); \theta^-) & \text{Otherwise.} \end{cases} \quad (8)$$

where r_j denotes the reward of the current state. γ represents the discount factor. s' denotes the next state, and θ denotes the neural network parameters of the current Q network. θ^- represents the neural network parameters of the target Q network.

The transfer matrix is input into the DDQN algorithm and the agent starts training to obtain the best attack path by the deep neural network, ϵ -strategy, network delay parameter update, experience replay and loss function calculation. As shown in Equation (9), the loss

function is the mean square error (MSE) of the difference between the current Q network value and the target Q network maximum value.

$$L(\theta_i) = \text{MSE}(Y_t^{\text{DDQN}} - Q(S_t, A_t; \theta)) \quad (9)$$

The agent updates the parameters of the target Q network after each certain number of training steps. Until the algorithm reaches the maximum number of training steps or invades the sensitive host, the agent reaches the end state. The Q network is updated as follows:

$$Q(S_t, A_t, \theta) \leftarrow Q(S_t, A_t, \theta) + \alpha[R_{t+1} + \gamma \max_a \hat{Q}(S_{t+1}, a, \theta) - Q(S_t, A_t, \theta)] \quad (10)$$

where α is the learning rate.

In this paper, the method of generating the transfer matrix is improved from Hu et al. [23]. That is, all nodes in the attack graph are mapped into a matrix that includes the common vulnerability scoring system version 3 (CVSS 3) [36] values of the vulnerabilities. It also includes other operations, such as accessing predefined scores for files.

For a clearer understanding of the MulVAL double deep Q network (MDDQN) algorithm, pseudocode is added, as shown in Algorithm 1.

Algorithm 1 MDDQN algorithm

```

1: Obtain target system topology information;
2: Set sensitive host value;
3: Use MulVAL to create attack tree;
4: Search attack graph paths using depth first algorithm;
5: Establish transfer matrix; /*Equation (6)*/
6: Neutral networks initialization;
7: for episode = 1 to maxepisode do
8:   for t = 1 to T do
9:     Use  $\epsilon$ -greedy to select action  $a$  according to Q value /*Equation (7)*/
10:    Execute action  $a$ , receive next state  $s'$ , reward  $r$ , and whether done
11:    Add transition tuple  $(s, a, r, s')$  to  $D$ 
12:     $s = s'$ 
13:    if  $|D| > \text{batchsize}$  then
14:      Sample and construct target Q value /*Equation (8)*/
15:      Do a gradient descent step with loss function /*Equation (9)*/
16:      Replace target parameters  $\theta^- \leftarrow \theta$  every N steps /*Equation (10)*/
17:    end for
18: end for

```

If the corresponding host h to the state s is not a sensitive host, the host value is 0 and the immediate reward $\text{Reward}(s, a)$ obtained by action a is negative. If the state corresponding to s' is a sensitive host, the immediate reward result is shown in Equation (11).

$$\text{Reward}(s, a) = \text{Reward}_{\text{Init}} - \text{Cost}_{\text{vul}} \quad (11)$$

$$\text{Reward}_{\text{Init}} = \text{Score}_{\text{vul}} + \text{Value}_h \quad (12)$$

where $\text{Reward}_{\text{Init}}$ represents the predefined scores of the vulnerability exploitation and privilege escalation in the transfer matrix, h represents all hosts invaded by the agent, and Cost_{vul} denotes the action cost of the vulnerability exploited by the agent.

The action cost is set to 6, 4, and 2 based on the high, medium, and low access complexity of the vulnerability common vulnerability system. As shown in Equation (12), $\text{Score}_{\text{vul}}$ is the CVSS 3 value of the vulnerability exploited by the agent, and Value_h denotes the asset value of host h . Based on these reward value settings, the goal of agent training is to attack the host with the most valuable host by using fewer operations.

4. Experiments

The experiments undertaken used Pytorch as the code structure of the algorithm. The hardware configuration included Intel i7-11800H CPU, 16G RAM, and Kali Linux as the operating system.

The experiments tested the effectiveness of the algorithm by building different scale penetration testing experimental scenarios. Firstly, different algorithms were used in the penetration testing experimental scenarios to compare the convergence speed at a certain number of training episodes. Then, in order to test the scalability of the algorithm, the MDDQN algorithm was used to train the agent in different scale penetration testing experimental scenarios.

4.1. Experiment Setup

The attack graph generated by Mulval is shown in Figure 3. The network shown in Figure 4 is scenario 1 of the penetration testing, which includes 3 subnets and 8 hosts. The subnets are separated by firewalls. The firewall only allows traffic that has been set in advance, and the host can communicate within the same subnet. The specific configuration of each host is shown in Table 2. The attacker performs vulnerability exploitation and privilege escalation operations based on the existing vulnerabilities and the running processes of the host.

In experimental scenario 1, for each host, the agent can perform vulnerability exploitation and privilege escalation operations. The vulnerabilities selected in the experiments are those frequently exploited by attackers during general penetration testing. For example, the agent can use the CVE-2021-3824 vulnerability for SQL injection and the CVE-2022-21510 vulnerability can be used to escalate privilege. The vulnerability information of the hosts is shown in Table 2. Each host defines the host value, the specific vulnerabilities, and the services and processes running on that host.

Table 2. Host configuration list.

Number	Host	Host Value	Vulnerability	Product	Protocol	Port
(1,0)	VPNServer	0	CVE-2021-3824	OpenVPN	SSL/HTTP	1194
(1,1)	CitrixServer	0	CVE-2017-6360	—	HTTP	8088
(2,0)	WebServer	0	CVE-2019-17571	httpd	TCP/UDP/HTTP	80
(2,1)	FileServer	0	CVE-2021-35223	—	FTP	21
(2,2)	Workstation	0	CVE-2022-24541	—	FTP/NFS/HTTP	8080
(3,0)	CommServer	0	CVE-2021-27085	IE	HTTP	9090
(3,1)	DatabaseServer	200	CVE-2022-21510	mounted	SSH/HTTP	1433
(3,2)	Operatingstation	0	—	—	HTTP	8088

In this paper, the proposed algorithm is tested and compared with other algorithms using three experimental scenarios. The number of subnets, hosts, and host vulnerabilities in each network environment is shown in Table 3 and the complexity of attack path planning gradually increases.

The MulVAL attack graph uses the Datalog language as the input text language. Therefore, in this paper, the penetration testing environment is described using the Datalog language. To verify the effectiveness of the algorithm, this paper analyzes the change in total reward with the number of training steps during the operation of the algorithm.

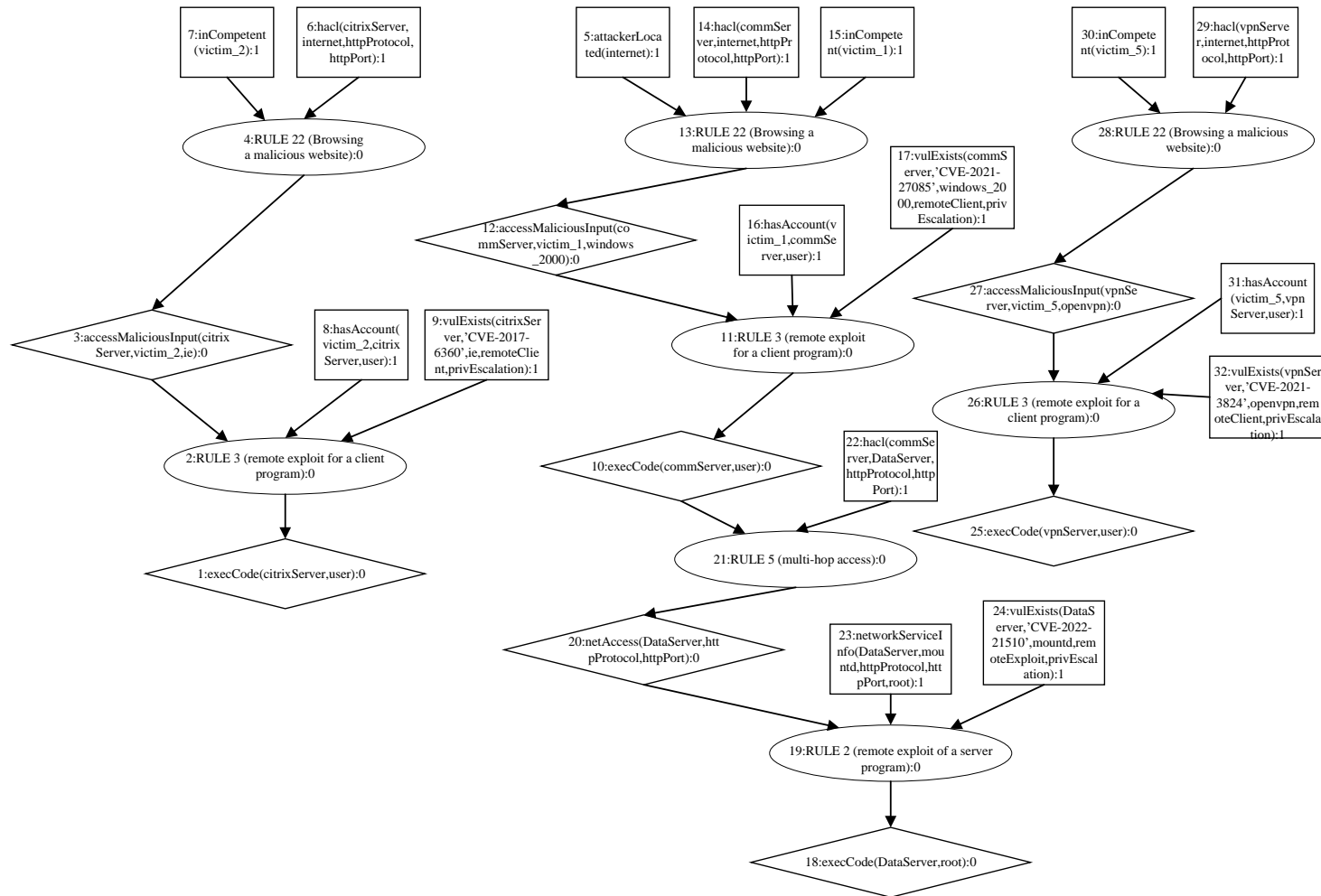


Figure 3. Attack graph.

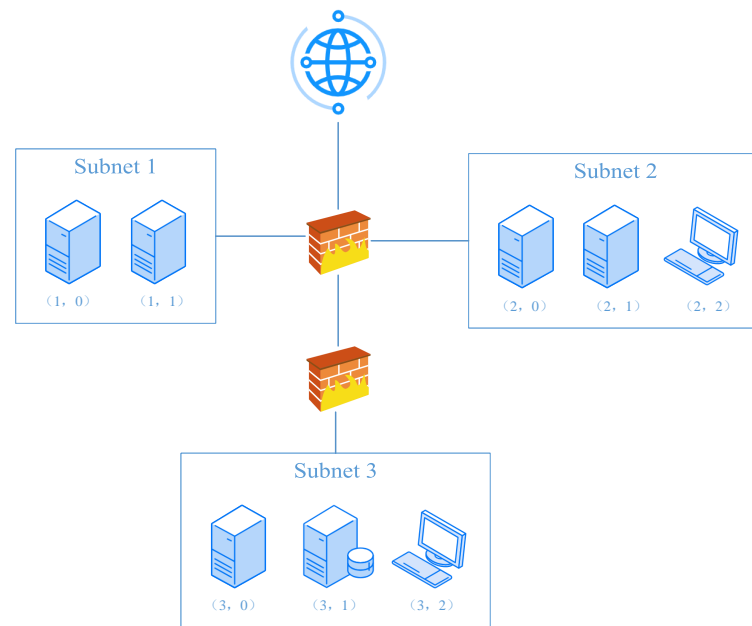


Figure 4. Penetration testing network environment in scenario 1.

Table 3. Experiment scenario list.

Scenario	Number of Subnets	Number of Hosts per Subnet	Number of Vulnerabilities	Accuracy Values	Average Number of Steps (MDDQN)
Scenario 1	3	[2,3,3]	7	190	3.78×10^5
Scenario 2	4	[2,4,3,4]	10	188	4.09×10^5
Scenario 3	6	[2,4,2,4,2,3]	14	188	5.17×10^5

4.2. Experimental Results and Analysis

The MDDQN algorithm proposed in this paper is compared with the DQN algorithm, the DDQN algorithm and the DuelingDQN algorithm. The experimental results are shown in Figure 5, which show as the number of training episodes grows, the changes in the rewards obtained by the agent for each algorithm in scenario 1. The hyperparameter setting is shown in Table 4.

Table 4. Hyperparameter list.

Hyperparameter	Value
Max steps	1,000,000
Max episode steps	2000
Learning rate	0.001
Batch size	32
Discount factor	0.9
Replay memory size	10,000
Hidden layer size	64
Target network update frequency	100

In scenario 1, four algorithms were tested, as shown in Figure 5. In scenario 1, all four algorithms reached convergence within a limited number of steps. Among them, the MDDQN algorithm converged the fastest, reaching convergence earlier than the other algorithms. Both the DQN algorithm and the DDQN algorithm showed large oscillations during training. The DDQN algorithm alleviated the problem of overestimation to a certain

extent relative to the DQN algorithm. Since the MDDQN algorithm has a priori knowledge provided for it by the attack graph, the DuelingDQN algorithm has a prioritized playback mechanism in it. Therefore, the MDDQN algorithm and DuelingDQN algorithm training process is more stable.

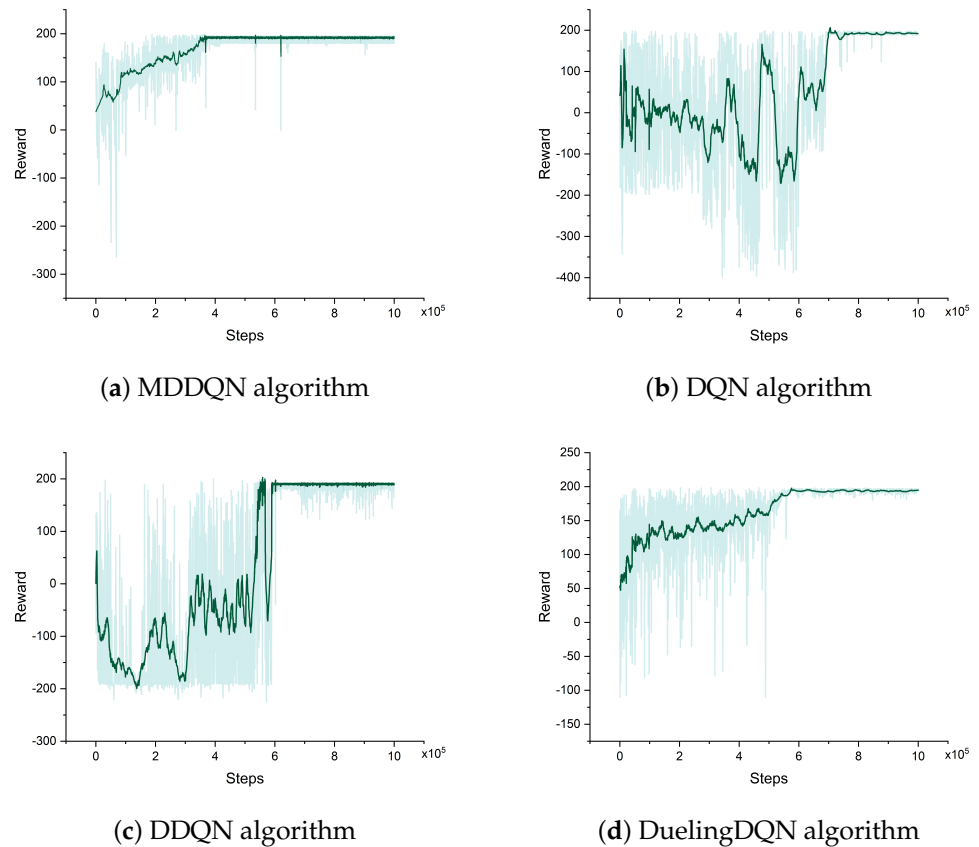


Figure 5. Algorithm experimental results in scenario 1.

To verify that the MDDQN algorithm has scalability, the algorithm was tested according to the list of experimental scenarios in Table 3. The experimental results are shown in Figures 6 and 7. In scenario 2, the maximum number of training steps in each episode was set to 2000 and the max steps was set to 2,000,000. In scenario 3, the maximum number of training steps in each episode was set to 5000 and the max steps was set to 2,000,000. The other hyperparameters were consistent with those listed in Table 3.

In scenario 2, as shown in Figure 6, the DQN algorithm failed to reach convergence within a limited step size. Due to the increased complexity of the experimental scenarios, both the DDQN algorithm and the DuelingDQN algorithm showed different degrees of oscillations during training. The DuelingDQN algorithm, due to the existence of the preferential return mechanism, obtained a higher reward value than the DDQN algorithm overall in training. The MDDQN algorithm, based on the priori knowledge provided by the attack graph, provides more positive incentives to the training, which enabled it to reach the convergence state quickly.

As shown in Figure 8, the fewer the number of steps per episode that the agent took to invade sensitive hosts, the closer the intelligent body was to the converged state. The four algorithms had a similar number of training steps per round at the beginning of training. As the intelligences continued to train iteratively, the MDDQN algorithm was the first to reach convergence.

As the complexity of the experimental scenario increased further in scenario 3 shown in Figure 7, the advantage of the MDDQN algorithm became more significant, with a more stable training process and the highest convergence efficiency.

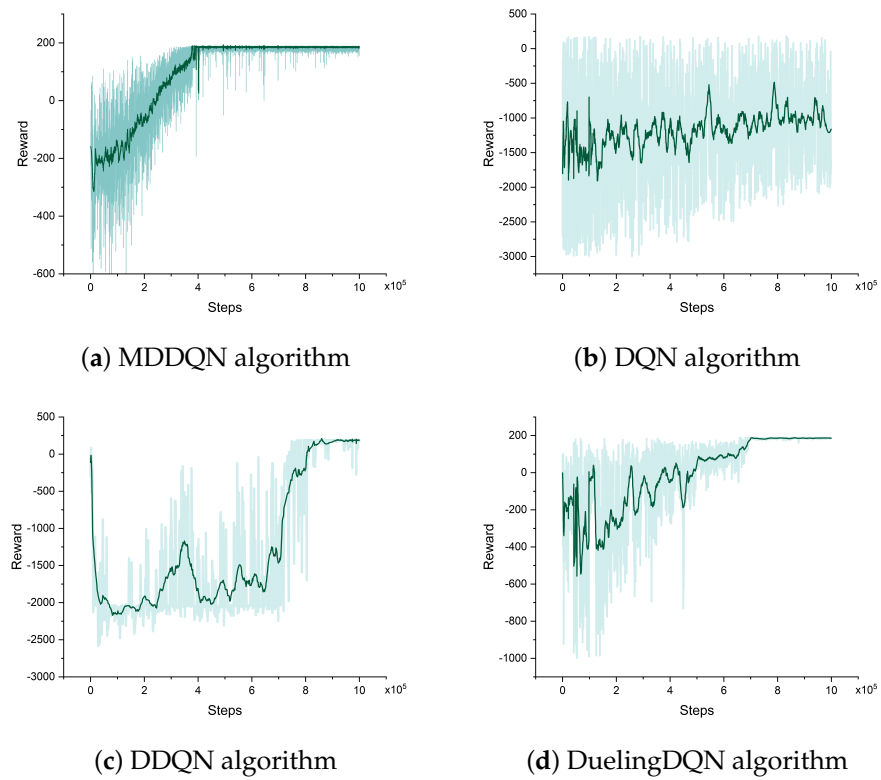


Figure 6. Algorithm experimental results in scenario 2.

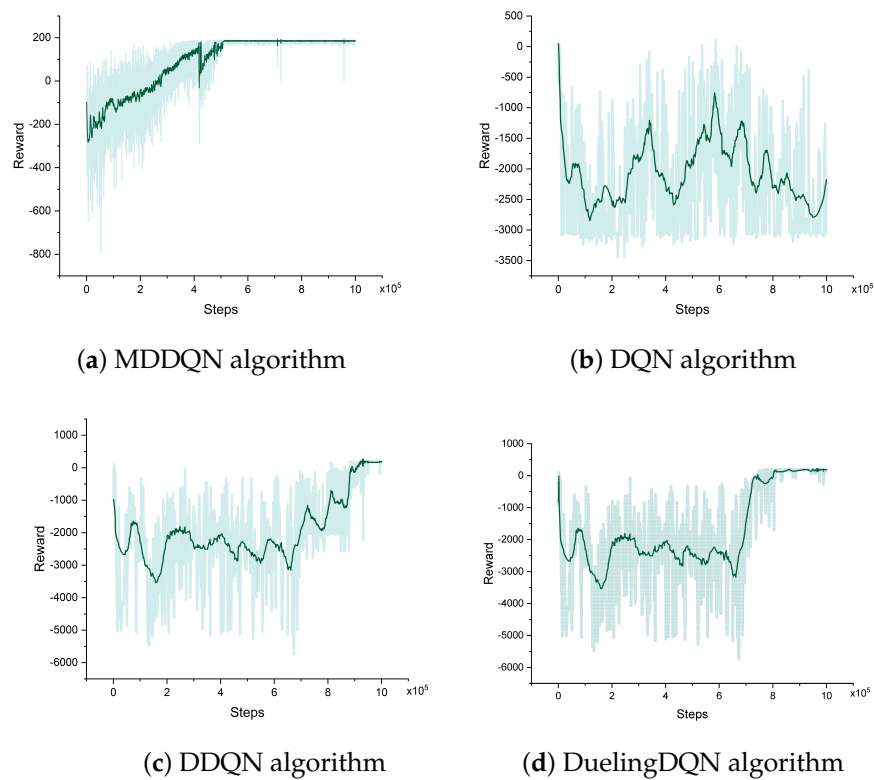


Figure 7. Algorithm experimental results in scenario 3.

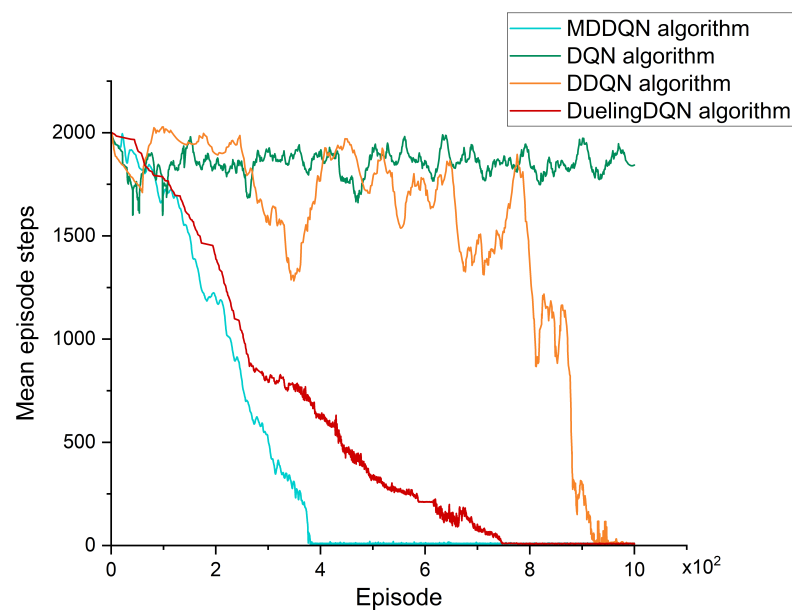


Figure 8. Mean episode steps versus episode.

In the MDDQN algorithm, the introduction of the attack graph improves the exploration efficiency of the intelligence, which helps the intelligence to learn the best strategy quickly in the large-scale state space. Also, the introduction of the DDQN algorithm makes the calculation of the Q value more accurate and improves the convergence speed and stability of the algorithm.

The MDDQN algorithm proposed in this paper solves the problem of sparse rewards in traditional DQN algorithms, that is, positive rewards only exist in a very small number of sensitive hosts in the whole network, and the vast majority of the agent's exploration often fails to obtain positive rewards. The transfer matrix generated from the attack graph results can give more positive guidance in the early stage of agent training, which improves the training speed of the agent. The introduction of the DDQN algorithm makes the Q value closer to the real value, and improves the convergence speed and the stability of the algorithm.

5. Conclusions and Future Work

In this paper, an MDDQN algorithm is proposed based on research relating to penetration testing attack path planning. The MulVAL attack graph is combined with the DDQN algorithm to provide more positive rewards to the DDQN algorithm by using the a priori knowledge provided by the attack graph. This solves, to some extent, the sparse reward problem prevalent in reinforcement learning algorithms. Then the MDDQN algorithm was subject to experimental testing for attack path planning. The experimental results showed that the MDDQN algorithm improved the convergence speed of the algorithm and the attack path planning efficiency was significantly improved. In order to verify the scalability of the algorithm in different scenarios, three sets of scenario experiments were set up. The experimental results showed that the advantages of the MDDQN algorithm were more obvious as the complexity of the experimental scenarios increased. Thus, compared with the other algorithms, the experimental results fully verify the superiority of the MDDQN algorithm in this paper. The MDDQN algorithm also suffers from certain limitations, such as an inability to autonomously scan the constructs and access network information, as well as a certain degree of overestimation.

In future studies, we intend to consider the problem of sparse rewards in more complex penetration testing environments and to increase the convergence speed of training. In order to improve the training efficiency of the agent, in future work, the setting method of the reward function, the calculation method of the Q value function, and the methods to guide the agent in the early stage of training will be changed.

Author Contributions: Conceptualization, X.L. and J.Y.; methodology, X.L. and J.Y.; software, X.L. and J.Y.; validation, X.L. and J.Y.; formal analysis, X.L. and J.Y.; investigation, J.Y.; resources, J.Y.; data curation, X.L. and J.Y.; writing—original draft preparation, X.L.; writing—review and editing, J.Y.; visualization, X.L. and J.Y.; supervision, J.Y.; project administration, J.Y.; funding acquisition, J.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

- McKinnel, D.R.; Dargahi, T.; Dehghantanha, A.; Choo, K.K.R. A systematic literature review and meta-analysis on artificial intelligence in penetration testing and vulnerability assessment. *Comput. Electr. Eng.* **2019**, *75*, 175–188. [\[CrossRef\]](#)
- Almazrouei, O.; Magalingam, P. The Internet of Things Network Penetration Testing Model Using Attack Graph Analysis. In Proceedings of the 2022 International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey, 20–22 October 2022; pp. 360–368.
- Wang, Z.; Zhang, Y.; Liu, Z.; Wei, X.; Chen, Y.; Wang, B. An automatic planning-based attack path discovery approach from IT to OT networks. *Secur. Commun. Netw.* **2021**, *2021*, 1444182. [\[CrossRef\]](#)
- Lamanna, L.; Saetti, A.; Serafini, L.; Gerevini, A.; Traverso, P. Online Learning of Action Models for PDDL Planning. In Proceedings of the IJCAI, Virtual, 19–26 August 2021; pp. 4112–4118.
- Liu, M.; Zhao, C.; Xia, J.; Deng, R.; Cheng, P.; Chen, J. PDDL: Proactive Distributed Detection and Localization Against Stealthy Deception Attacks in DC Microgrids. *IEEE Trans. Smart Grid* **2022**, *14*, 714–731. [\[CrossRef\]](#)
- Zhang, Y.; Liu, J.; Zhou, S.; Hou, D.; Zhong, X.; Lu, C. Improved Deep Recurrent Q-Network of POMDPs for Automated Penetration Testing. *Appl. Sci.* **2022**, *12*, 10339. [\[CrossRef\]](#)
- Ghanem, M.C.; Chen, T.M. Reinforcement learning for efficient network penetration testing. *Information* **2019**, *11*, 6. [\[CrossRef\]](#)
- Yang, S.; Mao, X.; Liu, W. Towards an extended pomdp planning approach with adjoint action model for robotic task. In Proceedings of the 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Toronto, ON, Canada, 11–14 October 2020; pp. 1412–1419.
- Li, Q.L.; Ma, J.Y.; Fan, R.N.; Xia, L. An overview for Markov decision processes in queues and networks. In *Stochastic Models in Reliability, Network Security and System Safety: Essays Dedicated to Professor Jinhua Cao on the Occasion of His 80th Birthday*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 44–71.
- Cody, T. A layered reference model for penetration testing with reinforcement learning and attack graphs. In Proceedings of the 2022 IEEE 29th Annual Software Technology Conference (STC), Gaithersburg, MD, USA, 3–6 October 2022; pp. 41–50.
- Van Hoang, L.; Nhu, N.X.; Nghia, T.T.; Quyen, N.H.; Pham, V.H.; Duy, P.T.; et al. Leveraging Deep Reinforcement Learning for Automating Penetration Testing in Reconnaissance and Exploitation Phase. In Proceedings of the 2022 RIVF International Conference on Computing and Communication Technologies (RIVF), Ho Chi Minh City, Vietnam, 20–22 December 2022; pp. 41–46.
- Gangupantulu, R.; Cody, T.; Park, P.; Rahman, A.; Eisenbeiser, L.; Radke, D.; Clark, R.; Redino, C. Using cyber terrain in reinforcement learning for penetration testing. In Proceedings of the 2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS), Barcelona, Spain, 1–3 August 2022; pp. 1–8.
- Hafiz, A. A Survey of Deep Q-Networks used for Reinforcement Learning: State of the Art. In *Intelligent Communication Technologies and Virtual Mobile Networks: Proceedings of ICICV 2022*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 393–402.
- Chowdhary, A.; Huang, D.; Mahendran, J.S.; Romo, D.; Deng, Y.; Sabur, A. Autonomous security analysis and penetration testing. In Proceedings of the 2020 16th International Conference on Mobility, Sensing and Networking (MSN), Tokyo, Japan, 17–19 December 2020; pp. 508–515.
- Chaudhary, S.; O'Brien, A.; Xu, S. Automated post-breach penetration testing through reinforcement learning. In Proceedings of the 2020 IEEE Conference on Communications and Network Security (CNS), Avignon, France, 29 June–1 July 2020; pp. 1–2.
- Wang, P.; Liu, J.; Zhong, X.; Yang, G.; Zhou, S.; Zhang, Y. DUSC-DQN: An Improved Deep Q-Network for Intelligent Penetration Testing Path Design. In Proceedings of the 2022 7th International Conference on Computer and Communication Systems (ICCCS), Wuhan, China, 22–25 April 2022; pp. 476–480.
- Tran, K.; Akella, A.; Standen, M.; Kim, J.; Bowman, D.; Richer, T.; Lin, C.T. Deep hierarchical reinforcement agents for automated penetration testing. *arXiv* **2021**, arXiv:2109.06449.
- Luong, N.C.; Hoang, D.T.; Gong, S.; Niyato, D.; Wang, P.; Liang, Y.C.; Kim, D.I. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 3133–3174. [\[CrossRef\]](#)
- Yang, Y.; Liu, X. Behaviour-diverse automatic penetration testing: A curiosity-driven multi-objective deep Reinforcement Learning approach. *arXiv* **2022**, arXiv:2202.10630.

20. Tran, K.; Standen, M.; Kim, J.; Bowman, D.; Richer, T.; Akella, A.; Lin, C.T. Cascaded reinforcement learning agents for large action spaces in autonomous penetration testing. *Appl. Sci.* **2022**, *12*, 11265. [\[CrossRef\]](#)
21. Cody, T.; Rahman, A.; Redino, C.; Huang, L.; Clark, R.; Kakkar, A.; Kushwaha, D.; Park, P.; Beling, P.; et al. Discovering exfiltration paths using reinforcement learning with attack graphs. In Proceedings of the 2022 IEEE Conference on Dependable and Secure Computing (DSC), Edinburgh, UK, 22–24 June 2022; pp. 1–8.
22. Ghanem, M.C.; Chen, T.M.; Nepomuceno, E.G. Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks. *J. Intell. Inf. Syst.* **2023**, *60*, 281–303. [\[CrossRef\]](#)
23. Hu, Z.; Beuran, R.; Tan, Y. Automated penetration testing using deep reinforcement learning. In Proceedings of the 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Genoa, Italy, 7–11 September 2020; pp. 2–10.
24. Mathew, A.; Amudha, P.; Sivakumari, S. Deep learning techniques: an overview. In *Advanced Machine Learning Technologies and Applications: Proceedings of AMLTA 2020*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 599–608.
25. Hooshmand, M.K.; Hosahalli, D. Network anomaly detection using deep learning techniques. *CAAI Trans. Intell. Technol.* **2022**, *7*, 228–243. [\[CrossRef\]](#)
26. Levine, S.; Kumar, A.; Tucker, G.; Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv* **2020**, arXiv:2005.01643.
27. Botvinick, M.; Ritter, S.; Wang, J.X.; Kurth-Nelson, Z.; Blundell, C.; Hassabis, D. Reinforcement learning, fast and slow. *Trends Cogn. Sci.* **2019**, *23*, 408–422. [\[CrossRef\]](#) [\[PubMed\]](#)
28. Arulkumaran, K.; Deisenroth, M.P.; Brundage, M.; Bharath, A.A. Deep reinforcement learning: A brief survey. *IEEE Signal Process. Mag.* **2017**, *34*, 26–38. [\[CrossRef\]](#)
29. Du, W.; Ding, S. A survey on multi-agent deep reinforcement learning: From the perspective of challenges and applications. *Artif. Intell. Rev.* **2021**, *54*, 3215–3238. [\[CrossRef\]](#)
30. Jang, B.; Kim, M.; Harerimana, G.; Kim, J.W. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access* **2019**, *7*, 133653–133667. [\[CrossRef\]](#)
31. Guo, S.; Zhang, X.; Du, Y.; Zheng, Y.; Cao, Z. Path planning of coastal ships based on optimized DQN reward function. *J. Mar. Sci. Eng.* **2021**, *9*, 210. [\[CrossRef\]](#)
32. Li, J.; Chen, Y.; Zhao, X.; Huang, J. An improved DQN path planning algorithm. *J. Supercomput.* **2022**, *78*, 616–639. [\[CrossRef\]](#)
33. Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; et al. Soft actor-critic algorithms and applications. *arXiv*, **2018**, arXiv:1812.05905.
34. Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In Proceedings of the AAAI conference on artificial intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
35. Tayouri, D.; Baum, N.; Shabtai, A.; Puzis, R. A Survey of MulVAL Extensions and Their Attack Scenarios Coverage. *IEEE Access* **2023**, *11*, 27974–27991. [\[CrossRef\]](#)
36. Nowak, M.; Walkowski, M.; Sujecki, S. Machine learning algorithms for conversion of CVSS base score from 2.0 to 3.x. In Proceedings of the Computational Science–ICCS 2021: 21st International Conference, Krakow, Poland, 16–18 June 2021; Proceedings, Part III; Springer: Berlin/Heidelberg, Germany, 2021; pp. 255–269.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.