International Journal for Housing Science and Its Applications

Publish September 9, 2025. Volume 47, Issue 1 Pages 236-247

https://doi.org/10.70517/ijhsa47120

Research on the Application of Deep Reinforcement Learning in Automated Penetration Testing Path Optimization and Decision Making

Wei Li[1], Zixuan Zhao[1,*], Youchen Shi[1], Yinquan Wang[1], Zeyang Zhao[1] and Hanlin Tu[1]

[1] Digital Intelligence Technology Company, PetroChina Xinjiang Oilfield Company, Karamay, Xinjiang, 834000, China

Corresponding authors: (e-mail: wlaqzxzzx@163.com).

Abstract This paper introduces deep reinforcement learning into automated penetration testing to plan and optimize penetration testing supply and defense paths. After modeling the automated penetration problem, the paper simplifies and evaluates the benefits of the DQN algorithm in deep reinforcement learning, finds the optimal penetration path through sample augmentation, and proposes the MASK-SALT-DQN algorithm. Through simulation experiments, the paper verifies the operation and effectiveness of the algorithm. In both simple and complex scenarios, the MASK-SALT-DQN algorithm achieves the fastest runtime speed, significantly enhancing the agent's learning efficiency. The algorithm provides accurate evaluation criteria for penetration testing path planning results. Compared to penetration testing learning algorithms based on Nature DQN, the MASK-SALT-DQN algorithm demonstrates a higher convergence value in its learning curve, indicating superior convergence performance.

Index Terms deep reinforcement learning, penetration testing, path optimization, MASK-SALT-DQN

I. Introduction

With the rapid deployment and adoption of fifth-generation mobile communications, networks have become one of the critical infrastructure components of the national economy, providing robust support for achieving universal connectivity. However, as a growing number of heterogeneous terminals and devices connect to networks, and new technologies such as network virtualization, software-defined networking, and edge computing are widely adopted, network boundaries are becoming increasingly blurred, expanding the network attack surface and making networks more vulnerable to attacks [1]-[4]. Cybersecurity incidents such as data breaches, ransomware, and hacking attacks are occurring with increasing frequency, and the economic losses caused by these incidents are also growing significantly [5]-[7]. To address cybersecurity risks, a series of security measures have been widely implemented, including conducting intrusion detection, deploying network firewalls and signaling firewalls, strengthening data encryption and integrity protection measures, and enhancing security isolation and secure transmission capabilities [8]-[11]. However, these methods primarily focus on the defender's perspective, passively detecting and maintaining the smooth operation of network systems, lacking proactivity in network protection [12]-[14]. With the emergence and evolution of highly stealthy unknown threats, an increasing number of researchers believe that cyberattacks are inevitable, and actively detecting network vulnerabilities and achieving prevention against cyberattacks holds significant importance [15], [16].

Penetration testing is an authorized simulated attack conducted on information systems to identify vulnerabilities that threaten their security, thereby assessing their security [17]. However, penetration testing techniques have their limitations. On one hand, the penetration testing process incurs significant time and economic costs, particularly when conducting penetration testing on large, complex network systems [18]-[20]. On the other hand, the results of penetration testing heavily depend on the cybersecurity knowledge level of the human experts conducting the testing, leading to significant variability [21]-[23]. As cyberattacks become increasingly frequent and complex, traditional manual penetration testing methods can no longer meet the demand for quickly identifying and addressing security vulnerabilities [24]. By automating penetration testing, it is possible to mitigate the human resource costs and reliance on human experts to some extent [25].

In recent years, deep reinforcement learning algorithms have been widely applied in the field of automated penetration testing. Reference [26] established an automated penetration testing framework based on deep reinforcement learning, which generates attack trees targeting specific network topologies and uses deep Q-learning networks (DQN) to screen for optimal

attack paths, providing effective solutions for defense training and attack training in cybersecurity activities. Reference [27] introduces reinforcement learning technology into cybersecurity activities to form an intelligent automated penetration testing system (IAPTS), which reduces the human resources and time costs of penetration testing by learning and reproducing complex penetration testing activities, while improving the frequency and reliability of testing. Reference [28] investigates automated test case generation methods in hardware verification, combining advanced deep reinforcement learning technology with traditional verification methods to achieve significant improvements in test coverage and security vulnerability detection capabilities. Literature [29] demonstrates that an automated penetration testing framework integrating deep reinforcement learning (DRL) methods exhibits excellent responsiveness, adaptability, and scalability, providing an effective solution for complex, dynamic, and high-dimensional network attack threats. Literature [30] models the network penetration testing process as a partially observable Markov decision process (POMDP) and proposes a new algorithm named ND3RQN to optimize the interaction strategy between the agent and the network environment, thereby identifying paths in the network structure that are more susceptible to attacks. Literature [31] also employs a deep Q-learning algorithm with a reward mechanism (DQRM) within a partially observable POMDP for automated penetration testing path planning. Experimental results indicate that knowledge-driven automated penetration testing methods based on reinforcement learning and reward mechanisms exhibit superior penetration testing performance. Reference [32] emphasizes that automated attack planning is the core concept of automated penetration testing. To this end, it proposes an automated attack planning (NIG-AP) algorithm based on network information acquisition, which can detect hidden attack paths from a hacker's perspective in POPDMP. Literature [33] provides a detailed analysis of relevant models, methods, and simulation environments for penetration path planning. Its application in automated penetration testing significantly improves the overall efficiency and success rate of detecting network and system vulnerabilities. Literature [34] addresses the requirements for efficient identification and rapid convergence in penetration testing within complex network environments by proposing an automated penetration testing model that combines curiosity mechanisms with reinforcement learning methods. In experimental simulations of network environments, this approach achieves both convergence speed and computational performance. In summary, automated penetration testing methods based on deep reinforcement learning can effectively reflect the uncertainty in the penetration process and are more suitable for penetration testing. However, they have disadvantages such as high computational complexity, slow convergence speed, high requirements for computational hardware, and unsuitability for large-scale networks. Therefore, reinforcement learning algorithms need to be specifically designed to optimize attack paths and decision-making.

The paper first analyzes penetration testing, discusses issues related to automated penetration testing, and models them. The most representative DQN algorithm in reinforcement learning is selected as the basis for the method in this paper. Based on the traditional DQN algorithm, after completing the solution space transformation, the solution is simplified and the benefits are evaluated. To accurately assess different vulnerabilities, the paper proposes the MASK-SALT-DQN algorithm by introducing a vulnerability exploitation sample enhancement method to increase the generation of relevant sample data during model training. To evaluate the performance of the proposed MASK-SALT-DQN algorithm, the paper analyzes its runtime, detection success rate, defense success rate, and algorithm hyperparameters through experiments.

II. Overview of Penetration Testing

Penetration testing is an active security assessment method that simulates an attacker's techniques to analyze security. Similar to malicious attacks, it also aims to discover confidential data in the system, gain control of the system, and identify potential risks that may affect business continuity. The biggest difference between the two is that penetration testing is a method of analyzing system security using non-destructive means, conducted with the approval of the client and under conditions of confidentiality. Through the risks identified during the penetration process, it provides solutions and security recommendations for the system.

II. A. Classification of Penetration Testing

Based on the scope of penetration testing, penetration testing can be divided into web penetration and internal network penetration. If classified according to different testing premises, penetration testing can also be divided into black box testing, white box testing, and gray box testing [35].

II. B. Steps in Penetration Testing

According to the Penetration Testing Standards (PTES), penetration testing can be divided into the following steps:

(1) Initial Interaction: The purpose of initial interaction is for penetration testers to communicate with the client to determine the scope, objectives, and constraints of the target system. At the same time, testers can obtain more information about the system under test from the client without significantly impacting the penetration testing process or results, thereby reducing the difficulty of their own testing. The client should consider factors such as the penetration testing timeline and effectiveness and provide appropriate assistance to the testers.

(2) Information Collection: During the information collection phase, testers will gather information about all assets within

the system based on its characteristics and the information provided by the client. This includes, from the outside in: website URLs, system IP addresses, internal network topology, system configurations, system devices, firewall status, and other security device information. Information collection is the most critical step in penetration testing. Testers can use network space engines, social engineering, information collection tools, and other methods to probe for sensitive information, thereby gaining a more comprehensive understanding of all aspects of the system under test, better grasping the key points of the test, reducing unnecessary attack attempts, and increasing the penetration success rate.

(3) Threat Modeling: Through early interaction and the information collection process, testers analyze potential security risks in the target network based on existing information and construct an overall penetration process (or penetration sequence) based on vulnerability exploitation methods. This process of developing an attack plan is referred to as threat modeling.

(4) Vulnerability Analysis: After developing the attack plan, testers must further steal sensitive information and control of the target system to conduct vulnerability analysis. Testers can analyze and summarize potential security issues based on previously summarized attack contexts, host information, vulnerability scanning results, and other factors, thereby facilitating deeper vulnerability discovery and identifying as many potential risks as possible.

(5) Vulnerability Exploitation: Vulnerability exploitation is the process by which testers verify security issues in the target system based on vulnerability analysis results. Testers must use actual vulnerability exploitation methods tailored to the type of vulnerability to initiate penetration against nodes in the target environment, with the aim of simulating an attacker's intrusion behavior to assess network robustness, steal sensitive information, or gain system privileges.

(6) Post-Penetration: In the post-penetration phase, testers must attempt to maintain persistent control over the system privileges obtained through vulnerability exploitation. Using existing device privileges as a foothold, they continuously penetrate other uncontrolled hosts and devices to conduct an in-depth analysis of the target network, ultimately gaining control over the entire system. This involves identifying all assets within the system that pose security threats and obtaining as much confidential information as possible to achieve better penetration results.

(7) Report Generation: The test report encompasses all critical intelligence information obtained by the tester, records the specific processes of detecting, identifying, and exploiting vulnerabilities, describes penetration sequences that could pose business threat risks, and outlines methods for obtaining sensitive information. Additionally, based on the aforementioned issues, the report proposes security maintenance recommendations and solutions.

III. Modeling automated penetration testing problems based on reinforcement learning

III. A. Description of automated penetration testing issues

III. A. 1) Automated Penetration Testing Problem Modeling

This paper is based on the characteristics of automated penetration testing scenarios and relevant knowledge in the field of cybersecurity. It combines formal modeling of automated penetration testing problems using open-source network attack and defense platforms such as CyberBattleSim to abstract the attack behavior of penetration testing entities as follows: This paper considers that penetration testing entities can perform three different types of attack actions: executing local attacks, executing remote attacks, and seizing control of nodes. A successful attack launched by a penetration testing entity may result in various outcomes, such as discovering new nodes, obtaining leaked credentials, extracting node attributes, seizing control of target nodes, or elevating privileges on target nodes. The penetration testing entity is equipped with a local vulnerability attack arsenal, a remote vulnerability attack arsenal, and a target port library, which are used to select local vulnerability attack weapons, remote vulnerability attack weapons, or target ports for use during attacks.

III. A. 2) POMDP Problem Modeling

This paper models the problem of automated penetration testing as a partially observable Markov decision process [36]. Formally, a partially observable Markov decision process can be represented by the tuple $\langle S, A, T, R, \Omega, O \rangle$. Here, S is the finite set of states of the target network system, Ω is the finite set of local observations of the target network system that the penetration tester can observe, A is the space of attack actions that the penetration tester can execute. At each time step t, the penetration tester is in a system state $s_t \in S$ and perceives a system local observation $o_t \in \Omega$. The penetration tester only has local observations of the target network system because they cannot predict all the information of the target network system in advance. For example, there are 100 host nodes in the target network system; at a certain moment, the penetration tester may have only discovered 13 host nodes in the system. The information about all 100 host nodes in the target network system and the network topology information constitute the environmental state, while the information about the host nodes discovered by the penetration testing entity constitutes the observation at the current time. The observation o_t perceived by the penetration testing entity depends on the probability distribution $o_t \sim O(s_t)$. Based on the perceived local observation, the penetration tester executes an attack $a_t \in A$ and transitions to the next state $s_t + 1$ according to the state transition probability distribution $T(s_t, a_t, s_{t+1})$ where $T(s_t, a_t, s_{t+1})$ is a conditional probability function, which can be expressed as $P(s_t + 1 \mid s_t, a_t)$ $R : S \times A \to R$ is the state-action-related reward function, which provides the immediate reward signal $r_t \sim R(s_t, a_t)$ obtained by taking each action in each state.

The solution to a partially observable Markov decision process problem is an optimal policy π^*. This optimal strategy maps the local observations of the penetration testing subject to the attack actions taken, with the objective of maximizing the expected long-term total reward $G_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau$, where $\gamma \in [0,1]$ is a discount factor that determines the relative importance of long-term rewards.

III. B. Overview of Reinforcement Learning

III. B. 1) Basic Concepts of Reinforcement Learning

Reinforcement learning is a machine learning paradigm in which an agent learns through trial and error, with behavior driven by rewards obtained through interaction with the environment. It is one of the three major machine learning paradigms, alongside supervised learning and unsupervised learning. As an important branch of machine learning, reinforcement learning differs significantly from the other two machine learning paradigms: unlike supervised learning, reinforcement learning does not rely on labeled training datasets but instead depends on the agent's own experience and learning capabilities; unlike unsupervised learning, the goal of reinforcement learning is to maximize long-term rewards rather than uncovering the underlying structure of the data. Reinforcement learning places greater emphasis on learning through interaction-oriented goals, acquiring knowledge and improving decision strategies through the agent's interaction with the environment. In reinforcement learning, the agent's goal is to maximize long-term cumulative rewards, i.e., returns, by selecting actions. The agent discovers which actions yield the greatest rewards solely through trial and error. The agent's actions not only influence immediate returns but may also affect the next environmental state, thereby impacting subsequent returns.

III. B. 2) Core Elements of Reinforcement Learning

(1) Strategy

A strategy defines the action guidelines that an intelligent agent should take in a specific environmental state. In essence, it maps the environmental state to the actions that the intelligent agent should perform. The strategy of an intelligent agent is typically represented by the symbol π. Based on the randomness of the strategy distribution, intelligent agents can be divided into two categories:

Deterministic strategy agents: The agent's strategy is expressed as $a = \pi(s)$. The agent can obtain a deterministic action a based on the strategy π and the current environmental state s.

Stochastic strategy agents: The agent's strategy is expressed as $\pi(a \mid s) = P(A_t = a \mid S_t = s)$. The agent can obtain the probability of executing action a based on the policy π and the current environment state s.

(2) Reward function

The reward function is a key component of reinforcement learning problems. It defines the short-term goal that the agent obtains after each action. In reinforcement learning, after the agent executes an action, the environment sends it a scalar value called a reward (which may be delayed). The reward represents the evaluation value of executing a specific action in a specific state.

Assuming that the agent takes action a in state s, the reward obtained by the agent can be expressed by equation (1):

$$R_s^a = E[R_{t+1} \mid S_t = s, A_t = a] \tag{1}$$

(3) Value function

The value function is mainly used to guide the intelligent agent in evaluating the merits of states or state actions from a long-term perspective. The value of a state is the expected cumulative (discounted) reward that the intelligent agent can obtain starting from that state.

Assuming that the strategy executed by the intelligent agent is π, the value function of the intelligent agent in state s can be expressed by equation (2):

$$v_\pi(s) = E_\pi(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s) \tag{2}$$

(4) Environment Model

An environment model is a simulation of external environmental behavior that can be used to infer the behavior of the environment. Based on a given state and action, the environment model predicts the probability distribution of the next state s' and the mathematical expectation of the next moment's reward R. This makes the environment model an important tool for planning.

III. B. 3) Deep Reinforcement Learning

As the overall state and actions in reinforcement learning continue to expand, using the aforementioned reinforcement learning solver algorithms becomes inefficient, and there is also the issue of conventional tables being unable to store large amounts of data. A feasible solution is to incorporate deep learning concepts into reinforcement learning, using neural networks to fit the action-value function $Q(s,a)$ of reinforcement learning, thereby replacing the traditional table-based data storage method of reinforcement learning algorithms. This special reinforcement learning solver method is referred to as deep reinforcement learning [37].

DQN is the most representative algorithm in deep reinforcement learning [38]. Its core idea is to combine convolutional neural networks (CNNs) with Q-learning in reinforcement learning, using three convolutional layers and two fully connected layers to perform nonlinear transformations. The output layer then generates the value of $Q(s,a)$ for each state-action pair and calculates the current policy. DQN typically uses an actor-learner architecture, where the DQN-actor handles interaction with the environment, and the DQN-learner updates the action-value function and policy.

To address issues such as instability when representing value functions with nonlinear networks, DQN employs two neural networks to represent the current Q-value and the target Q-value (target Q). The current Q-value is used to estimate the reward obtained during action exploration, and after every N steps, the parameters of the estimation network are synchronized to the target network. The target Q-value is used to generate the optimal policy. Additionally, DQN stores samples of interactions with the environment ($S_t, A_t, reward, S_{t+1}$, i.e., actions, resulting state transitions, and rewards) in an experience pool (replay buffer). After a certain number of action steps, it randomly selects samples from the experience pool for learning:

$$Y_i = r + \gamma maxQ(s^{'}, a^{'} \mid \theta_{i'}) \tag{3}$$

Among them, Y_i is the target Q value.

Then, the gradient descent algorithm is used to update the neural network parameters θ, whose loss function and gradient function are respectively:

$$L(\theta_i) = E_{s,a,r,s'}[(Y_i - Q(s,a \mid \theta_i))\hat{}\,2] \tag{4}$$

$$\nabla_{\theta_i} L(\theta_i) = E_{s,a,r,s'}[(Y_i - Q(s,a \mid \theta_i)\nabla_{\theta_i} Q(s,a \mid \theta_i)] \tag{5}$$

Among them, $L(\theta_i)$ is the loss function, and ∇_{θ_i} is the gradient.

IV. Automated penetration path planning and optimization

IV. A. Simplification and Benefit Assessment

The DQN algorithm selects the next action based on the expected cumulative future reward that each action in the action list can bring after execution, as evaluated from the current state. The DQN algorithm consists of two networks: eval_network and target_network. The agent first observes the state s and calculates the expected future reward for each action using eval_network, i.e., the Q-value. Then, according to the ε greedy rule to select action a to execute, obtains reward r and observes new state $s^{'}$, iterating this process to obtain a series of quadruples (s,a,r,s') and storing them in the experience replay pool. In each round, the model selects a set of quadruples from the experience replay pool and calculates the loss based on the mean squared error. For each selected quadruple (s,a,r,s'), the target_network calculates the value of each action under $s^{'}$ and selects the maximum Q-value for calculating the target Q-value. Additionally, the parameters of the eval_network are copied to the target_network at regular intervals. The training process of DQN is shown in Figure 1.

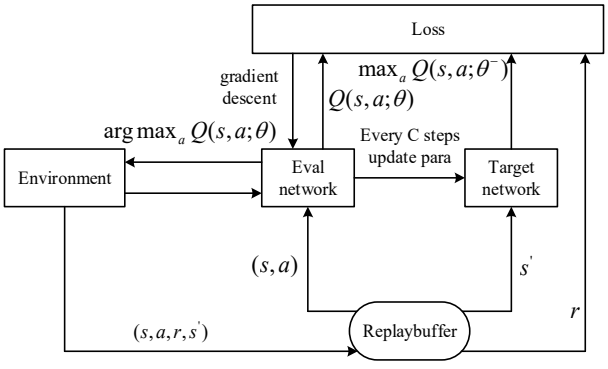


Figure 1: DQN training process

The calculation formula for the target Q value is shown in Equation (6):

$$y_i = r + \gamma \max_{a'} Q(s', a'; \theta_i^-) \tag{6}$$

The calculation formula for loss is shown in Equation (7):

$$L_i(\theta_i) = E_{(s,a,r,s')\sim D}[(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))\nabla_{\theta_i} Q(s, a; \theta_i)] \tag{7}$$

Among them, D is the empirical replay pool; θ_i is the parameter of eval_network; y_i is the target Q value; γ is a discount factor between 0 and 1, used to balance the importance of immediate rewards and future rewards.

Then, gradient descent is used for gradient updating. The gradient formula of loss is shown in Equation (8):

$$\nabla_{\theta_i} L_i(\theta_i) = E_{(s,a,r,s')\sim D}[(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))\nabla_{\theta_i} Q(s, a; \theta_i)] \tag{8}$$

The solution space transformation based on invalid action masking is achieved by modifying the action selection process in the above model. In the eval_network, invalid action masking is divided into two cases. In the first case, the model randomly selects actions with a probability of ε. To exclude the selection of invalid actions, it is first necessary to obtain a list of valid actions based on the current state, and then limit the selection range to that list. In the second case, the model selects the action with the maximum Q-value with a probability of $1-\varepsilon$. This requires obtaining the list of invalid actions based on the current state s, then replacing the Q-values of the invalid actions with the minimum value $M(-inf)$, and finally allowing the model to select the action with the maximum Q-value. The target_network always implements masking according to the second case described above.

Next, we analyze the effectiveness of the solution space transformation in the above model processing. In each step of DQN training, the above solution space transformation process covers all cases of action selection, ensuring that the model does not select invalid actions. As can be seen from the loss function described in Equation (6), the Q-values involved in calculating the loss are the Q-value of action a selected by the eval_network in the current state s and the Q-value of action a' selected by the target_network in the transitioned state s'. Since the solution space transformation has been performed, both actions a and a' are valid actions. Therefore, the calculated loss and its gradient are not affected by invalid actions, meaning that invalid actions do not participate in the model's iterative update process. In summary, the solution space transformation method proposed in this paper ensures that the agent does not select invalid actions during the exploration process while also not affecting the model's update process.

As described above, during the model exploration process, the first positive reward obtained for reaching the goal results in a large TD-error for that transition, significantly affecting the model adjustment. Solution space transformation allows the model to select actions only from the list of valid action candidates, and valid actions have a higher probability than invalid actions of transitioning the state toward the goal state. Therefore, the masked model can reach the goal in fewer steps during the initial exploration phase. Let the model with solution space transformation and the general model be denoted as M_1 and M_2, respectively. Assuming that M_1 requires an average of n steps to initially reach the target state, let M_1 is the proportion of invalid actions at the i th exploration step, where $\beta_i, 0 < \beta_i < 1$. Then, the random variable X_i describes the number of steps required for M_2 to explore valid actions compared to M_1 at this stage. The distribution function $F(X_i)$ of X_i is shown in Equation (9):

$$F(x) = P\{X_i \le x\} = \sum_{j=1}^{x} \beta_i^{j-1}(1-\beta_i) \tag{9}$$

Then, the sum of the number of steps required for M_2 to first explore the target state is the sum of $X_1, X_2, \cdots, X_n$, as shown in Equation (10). The expected value of E_{step} represents the average number of steps required for M_2 to successfully explore the target state for the first time. The expected value calculation is shown in Equation (11):

$$sum = \sum_{1\le i\le n} X_i \tag{10}$$

$$E_{step} = E(sum) = \sum_{1\le i\le n} E(X_i) = \sum_{1\le i\le n} \frac{1}{1-\beta_i} \tag{11}$$

It can be seen that during the initial exploration phase of M_1, when the average number of steps to reach the target state is n, M_2 without solution space transformation requires far more than n steps, as indicated by E_{step}. Therefore, solution space transformation effectively reduces the number of steps required to achieve the target state for the first time. To quantify the effect of the solution space transformation, the compression ratio η is calculated using the expected number of steps required for the two models to first reach the target, as shown in Equation (12):

$$\eta = \frac{E_{step}}{n} \tag{12}$$

The higher the compression ratio, the faster the agent achieves successful exploration on its first attempt through the solution space transformation, and the faster the model adjustment speed. Additionally, as the network scale increases, the proportion of ineffective actions during the initial exploration phase, denoted as β_i, becomes larger, resulting in a higher compression ratio.

However, in the actual model training process, there is a step limit for each round of exploration, denoted as N. When the model's exploration steps exceed N, it will restart the exploration. Therefore, it is necessary to make certain corrections to sum. Assuming that the probability of $sum < N$ is p, the distribution function $R(S)$ of the number of rounds of model exploration S is shown in Equation (13):

$$R(s) = P\{S \le s\} = \sum_{1 \le j \le s} (1-p)^{j-1} p \tag{13}$$

When the number of exploration rounds is s, the number of exploration steps should be $N*(s-1)+m'$, where m' has the same distribution as sum, but is restricted to $m' < N$. In summary, the expected number of steps E'_{step} required for the first exploration of the corrected M_2 to complete the objective is calculated by equation (14):

$$\begin{aligned} E'_{step} &= E_s(N*(s-1)) + E(m') \\ &= \sum_{s=1}^{\infty} P\{S=s\}*(s-1) + E(m') \\ &= \frac{1-p}{p} N + E(m') \end{aligned} \tag{14}$$

IV. B. Sample Enhancement

Accurate assessment of different vulnerabilities is key to finding the optimal penetration path. At any given moment, an agent has multiple potential attack targets and vulnerabilities, and the uncertainty of the results of vulnerability exploitation actions is significant. The agent requires more sample data to accurately assess and select appropriate vulnerability exploitation actions. However, during model training, especially in the early stages, action selection primarily relies on random exploration by the model, and the probability of different actions being selected is the same. Therefore, the number of samples related to vulnerability exploitation actions does not have an advantage. Additionally, as the network scale expands, the probability of hosts in the deeper parts of the network being explored decreases, leading to a scarcity of vulnerability exploitation-related samples for targets and insufficient evaluation of corresponding actions, resulting in an unstable model convergence process. Therefore, the model requires a large number of training steps to converge to the optimal penetration path. The solution process can be optimized by improving the composition of data in the experience replay pool.

By proposing a vulnerability exploitation sample enhancement method (MASK-SALT-DQN), the generation of vulnerability exploitation action-related sample data during model training can be increased. When the model selects an exploit-type action, the agent first obtains all exploit actions that can be executed on the action target, then repeats these actions several times. At the same time, to prevent the generation of too many exploit-related samples, which would reduce the probability of sampling other action-related samples and further affect the evaluation of these actions and the convergence of the model, a certain threshold needs to be set. When the model's exploration rate falls below this threshold, vulnerability exploitation sample augmentation is no longer performed, and the number of samples generated can be reduced, thereby shortening the training time. Additionally, since vulnerability exploitation samples for hosts in the deeper layers of the network are relatively scarce, to prevent an imbalance in the proportion of vulnerability exploitation samples across different hosts, greater emphasis should be placed on enhancing vulnerability exploitation samples for hosts in the deeper layers of the network.

V. Experiments and analysis of results

V. A. Algorithm Performance Analysis

This paper uses a network simulator to design a penetration testing environment. It demonstrates changes in total rewards and runtime during algorithm execution to validate algorithm performance. Three network environments of different scales are first designed to test the performance of the algorithm proposed in this paper against other algorithms. The number of subnets, the number of hosts per subnet, and the number of host vulnerabilities in each network environment gradually increase, while the complexity of the network structure also increases, requiring attackers to take more attack steps to reach the target state.

The experimental hardware configuration includes an Intel i9-7980XE CPU, 128GB of memory, and the Windows 10 operating system. The algorithm program is written in Python. The hyperparameter values for the three experimental scenarios

are shown in Table 1.

Table 1: The settings of the lower points of different experimental scenarios

	MAX_STEPS	MAX_EP_STEPS	EXPLORATION_STEPS	START_STEPS	REPLAY_SIZE
Scenario 1	80000	400	8000	3000	8000
Scenario 2	80000	800	40000	6000	8000
Scenario 3	120000	4000	65000	6000	15000

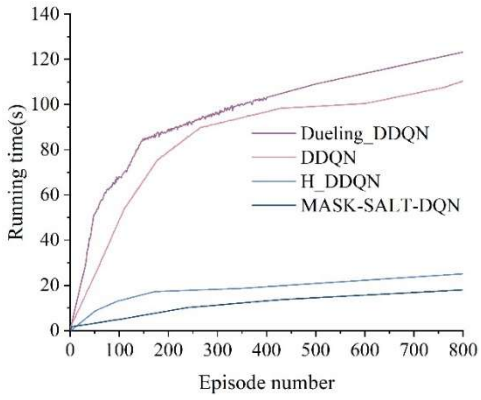
V. A. 1) Operating time

First, we compare the DDQN algorithm and Dueling_DDQN. The DDQN algorithm with path heuristic information is denoted as H_DDQN, and the algorithm proposed in this paper is denoted as MASK-SALT-DQN. Experiments are conducted under consistent hyperparameter and environment settings. All four algorithms eventually converge to the optimal environment reward value. The time changes during the execution of each algorithm in the three scenarios are shown in Figure 2.

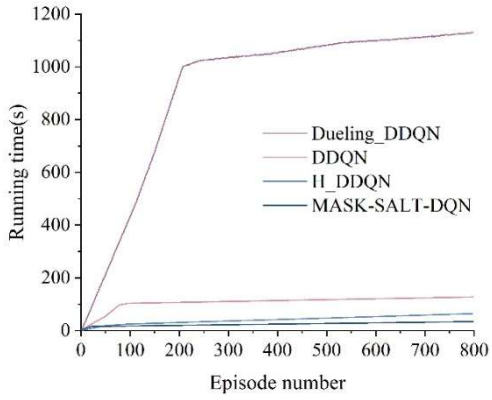
From the experimental results in Scenario 1, it can be seen that under small-scale experimental conditions, the runtime of the DDQN and Dueling_DDQN algorithms is similar, while the runtime of the H_DDQN and MASK-SALT-DQN algorithms is similar, both significantly outperforming the first two algorithms, indicating that the proposed algorithm can improve the agent's learning speed.

From the experimental results in Scenario 2, it can be seen that after the network scale is expanded, the Dueling_DDQN algorithm experiences rapid runtime growth and slow convergence in test environments with large state spaces. The MASK-SALT-DQN algorithm has the fastest runtime, with the time required to reach convergence being approximately 18 seconds.

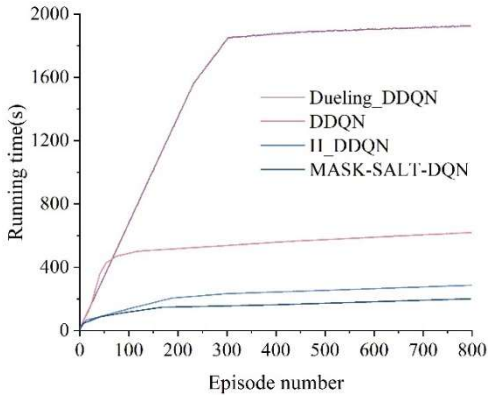
From the results of Scenario 3, it can be seen that the H_DDQN and MASK-SALT-DQN algorithms are more efficient in the learning process compared to the other two algorithms, and as the problem scale expands, the improvement of the MASK-SALT-DQN algorithm compared to other algorithms becomes more pronounced.



(a) Running time in scenario 1



(b) Running time in scenario 2



(c) Running time in scenario 3

Figure 2: Four algorithm training process running time

V. A. 2) Detection Success Rate and Defense Success Rate

Design a penetration testing scenario for simulation. In addition to the attacker, there are a total of five hosts in the network, two of which are servers. The server in LAN1 provides web services, while the server in LAN2 provides file services. The attacker's goal is to obtain root privileges on the file server in LAN2. The attacker has root privileges on their own host.

Next, based on the above network connection and host configuration information, we will outline the actions taken by the attacker and the defender. The relevant action information of the attacker and the defender, the attacker action number A1~A7, the defender action number D1~D10, define the asset values for different asset attributes of the network entity, the asset values of the web server and the file server are (20, 10, 10, 20, 30) and (20, 30, 10, 30, 10, 10, 20), and the rest of the hosts are (10, 10, 10, 10, 10, 10). Regarding the importance of network entities within the network, two servers are set to 4, and the remaining servers are set to 2. The preference values for different assets are set to (0,1,1,1,1,1) and (1,1,0,1,1,1) for the attacking and defending sides, respectively. The detection success rate and defense success rate after detection of the algorithm in this paper are shown in Table 2.

Table 2: Test success rate and test success rate

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d_j
A1	0.78	0	0	0	0	0	0	0.88	0	0	0.8
A2	0	0.96	0	0	0	0	0.12	0	0.24	0	0.7
A3	0.06	0.14	0.82	0.47	0	0	0	0	0	0	0.4
A4	0	0.2	0.07	0.75	0	0	0	0	0.18	0	0.5
A5	0	0	0.33	0	0.78	0	0	0	0.13	0	0.3
A6	0.15	0	0	0.28	0	0.72	0.86	0	0	0	0.8
A7	0	0	0	0.16	0	0	0	0	0.83	0.92	0.6

Calculate the offensive and defensive utility of the attacker and defender in each action round. The results are shown in Tables 3 and 4. Assuming there are two attack paths, A1 and A2, the defender has 12 possible defensive strategies, denoted as D1 to D12.

Next, the optimal attack and defense strategies are selected using the expected utility matrix under pure strategies. From the above settings, it can be seen that the attack action a2 used in the second attack path has a lower cost, while the corresponding defense action d2 has a higher defense cost, making the second attack path more feasible than the first attack path. This indicates that the attack path evaluation method proposed in this paper is reasonable and can provide a relatively accurate evaluation criterion for the attacker's path planning results.

Table 3: Attack returns

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
a1	−45.25	−4	−4	−4	−4	−4	−4	−52.47	−4	−4
a2	5	−44.12	5	5	5	5	−1.25	5	−12.04	5
a3	28.63	27.46	13.56	21.17	32	32	32	32	32	32
a4	−25	−30.42	−26.84	−48.52	−25	−25	−25	−25	−32.06	−25
a5	150	150	135.22	150	115.42	150	150	150	142.67	150
a6	28.54	32	32	22.63	32	3.48	−2.55	32	32	32
a7	205	205	205	184.23	205	205	205	205	105.52	92.78

Table 4: Defensive returns

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
a1	30.4	−84	−18	−36	−55	−18	−55	3	−18	−36
a2	−18	−12	−18	−36	−55	−18	−50	−55	82.6	−36
a3	−16.2	−85	5	−25.3	−55	−18	−55	−55	−18	−36
a4	−18	−78	−15	2	−55	−18	−55	−55	30.6	−36
a5	−18	−84	−12	−36	5	−18	−55	−55	−12.2	−36
a6	−10.5	−84	−18	−20.2	−55	32	682.6	−55	−18	−36
a7	−18	−84	−18	−30.5	−55	−18	−55	−55	152	155

V. B. Algorithm Hyperparameter Analysis

For the penetration testing learning algorithm described in this paper, when the discount factor γ is set to 0.99 and the learning step size α is set to 0.01, 0.05, and 0.1, respectively, the returns obtained are shown in Figure 3. The vertical axis represents the returns obtained by the algorithm in each round, and the horizontal axis represents the number of training rounds. All figures in this section are set up in the same way. Although the algorithm converges stably in all three cases, there are subtle differences. When α = 0.05 and α = 0.1, the distribution of return values in later training rounds is closer to 17. When α = 0.01, the fitted curve is smoother, and the convergence is better than when α = 0.05 and α = 0.1.

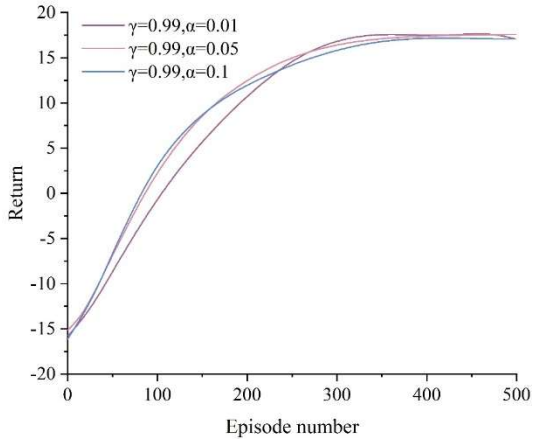


Figure 3: Learning curve

Using the Nature DQN algorithm as a comparison method, we analyze the impact of batch size on the learning performance of the penetration testing learning algorithm based on Nature DQN. At this point, the learning step size α is set to 0.01, the discount factor γ is set to 0.95, and the update frequency is set to update the target Q network every 15 iterations. Then, the batch size is sequentially set to 128, 256, and 512. As the batch size increases, the learning curve of the algorithm gradually rises. However, when the batch size is 512, the algorithm does not converge stably.

To analyze the impact of learning step size on the learning performance of the penetration testing learning algorithm based on Nature DQN. At this point, the batch size is set to 128, the discount factor γ is set to 0.95, and the update frequency is set to update the target Q-network every 15 rounds. Then, the learning step size α is set to 0.01 and 0.001, respectively. By comparing the results, it can be seen that the learning performance of the algorithm with a learning step size of 0.01 is superior to that with a learning step size of 0.001.

To analyze the impact of the discount factor size on the learning performance of the penetration testing learning algorithm based on Nature DQN. At this point, the batch size is set to 512, the update frequency is set to update the target Q-network every 15 rounds, and the learning step size α is set to 0.01. Then, the discount factor γ is set to 0.85 and 0.99, respectively. The learning curves corresponding to these two cases are shown in Figure 4. By comparison, it can be seen that when the discount factor is close to 1, the learning performance of the algorithm decreases.

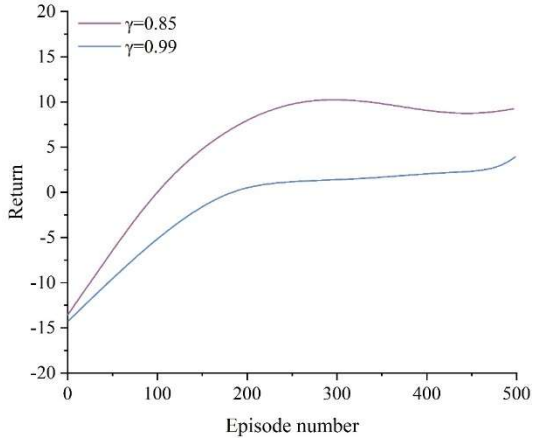


Figure 4: Learning curve when γ=0.85, 0.99

In summary, for the penetration testing learning algorithm based on Nature DQN, to achieve high learning efficiency, the update frequency should be set to 15, the batch size to 256, the learning step size to 0.01, and the discount factor to 0.95.

When comparing the penetration testing learning algorithm proposed in this paper with the penetration testing learning algorithm based on Nature DQN, the former exhibits a higher convergence value in its learning curve, while the latter shows signs of non-convergence toward the end of training.

VI. Conclusion

This paper applies deep reinforcement learning algorithms to automated penetration testing, performing path planning and optimization for supply and defense paths. Through experiments, the performance and hyperparameters of the deep reinforcement learning algorithms proposed in this paper are analyzed.

In three scenarios, the runtime of the MASK-SALT-DQN algorithm proposed in this paper is shorter than that of other algorithms, improving the learning speed of the agent. The second attack path planned in this paper is more feasible, and the attack path evaluation method is reasonable, providing an accurate standard for assessing attack path planning results. When the discount factor $\gamma = 0.99$ and $\alpha = 0.05$, the fitted curve is relatively stable, with better convergence. The convergence value of the learning curve for the MASK-SALT-DQN penetration testing learning algorithm in this paper is higher than that of the penetration testing learning algorithm based on Nature DQN.

References

[1] Paráda, I. (2018). Basic of cybersecurity penetration test. Hadmérnök, 13(3), 435-442.

[2] Bin Arfaj, B. A., Mishra, S., & AlShehri, M. (2022). Efficacy of Unconventional Penetration Testing Practices. Intelligent Automation & Soft Computing, 31(1).

[3] Bella, G., Biondi, P., Bognanni, S., & Esposito, S. (2023). Petiot: Penetration testing the internet of things. Internet of Things, 22, 100707.

[4] Al-Hawamleh, A. M. (2023). Predictions of cybersecurity experts on future cyber-attacks and related cybersecurity measures. momentum, 3, 15.

[5] Igwenagu, U. T. I., Salami, A. A., Arigbabu, A. S., Mesode, C. E., Oladoyinbo, T. O., & Olaniyi, O. O. (2024). Securing the digital frontier: Strategies for cloud computing security, database protection, and comprehensive penetration testing. Journal of Engineering Research and Reports, 26(6), 60-75.

[6] Lachkov, P., Tawalbeh, L. A., & Bhatt, S. (2022). Vulnerability assessment for applications security through penetration simulation and testing. Journal of Web Engineering, 21(7), 2187-2208.

[7] Formosa, P., Wilson, M., & Richards, D. (2021). A principlist framework for cybersecurity ethics. Computers & Security, 109, 102382.

[8] Al-Zadjali, B. M. (2016). Penetration testing of vulnerability in android Linux kernel layer via an open network (Wi-Fi). International Journal of Computer Applications, 975(8887).

[9] Waraga, O. A., Bettayeb, M., Nasir, Q., & Talib, M. A. (2020). Design and implementation of automated IoT security testbed. Computers & security, 88, 101648.

[10] Pratama, D., Suryanto, N., Adiputra, A. A., Le, T. T. H., Kadiptya, A. Y., Iqbal, M., & Kim, H. (2024). Cipher: Cybersecurity intelligent penetration-testing helper for ethical researcher. Sensors, 24(21), 6878.

[11] Dalalana Bertoglio, D., & Zorzo, A. F. (2017). Overview and open issues on penetration test. Journal of the Brazilian Computer Society, 23(1), 2.

[12] Goel, J. N., & Mehtre, B. M. (2015). Vulnerability assessment & penetration testing as a cyber defence technology. Procedia Computer Science, 57, 710-715.

[13] McKinnel, D. R., Dargahi, T., Dehghantanha, A., & Choo, K. K. R. (2019). A systematic literature review and meta-analysis on artificial intelligence in penetration testing and vulnerability assessment. Computers & Electrical Engineering, 75, 175-188.

[14] Shah, S., & Mehtre, B. M. (2015). An overview of vulnerability assessment and penetration testing techniques. Journal of Computer Virology and Hacking Techniques, 11(1), 27-49.

[15] Altulaihan, E. A., Alismail, A., & Frikha, M. (2023). A survey on web application penetration testing. Electronics, 12(5), 1229.

[16] Nagpure, S., & Kurkure, S. (2017, August). Vulnerability assessment and penetration testing of web application. In 2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA) (pp. 1-6). IEEE.

[17] Sarker, I. H., Kayes, A. S. M., Badsha, S., Alqahtani, H., Watters, P., & Ng, A. (2020). Cybersecurity data science: an overview from machine learning perspective. Journal of Big data, 7(1), 41.

[18] Schwartz, J., Kurniawati, H., & El-Mahassni, E. (2020, June). Pomdp+ information-decay: Incorporating defender's behaviour in autonomous penetration testing. In Proceedings of the International Conference on Automated Planning and Scheduling (Vol. 30, pp. 235-243).

[19] Samad, A., Altaf, S., & Arshad, M. J. (2024). Advancements in Automated Penetration Testing for IoT Security by Leveraging Reinforcement Learning. evaluation, 8, 9.

[20] Shaukat, K., Luo, S., Varadharajan, V., Hameed, I. A., & Xu, M. (2020). A survey on machine learning techniques for cyber security in the last decade. IEEE access, 8, 222310-222354.

[21] Handa, A., Sharma, A., & Shukla, S. K. (2019). Machine learning in cybersecurity: A review. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 9(4), e1306.

[22] Ghaffarian, S. M., & Shahriari, H. R. (2017). Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. ACM computing surveys (CSUR), 50(4), 1-36.

[23] Sarker, I. H., Abushark, Y. B., Alsolami, F., & Khan, A. I. (2020). Intrudtree: a machine learning based cyber security intrusion detection model. Symmetry, 12(5), 754.

[24] Semenov, S., Weilin, C., Zhang, L., & Bulba, S. (2021). Automated penetration testing method using deep machine learning technology. Advanced Information Systems, 5(3), 119-127.

[25] Alqahtani, F. H., & Alsulaiman, F. A. (2020). Is image-based CAPTCHA secure against attacks based on machine learning? An experimental study. Computers & Security, 88, 101635.

[26] Hu, Z., Beuran, R., & Tan, Y. (2020, September). Automated penetration testing using deep reinforcement learning. In 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW) (pp. 2-10). IEEE.
[27] Ghanem, M. C., & Chen, T. M. (2019). Reinforcement learning for efficient network penetration testing. Information, 11(1), 6.
[28] Chen, J., Yan, L., Wang, S., & Zheng, W. (2024). Deep reinforcement learning-based automatic test case generation for hardware verification. Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023, 6(1), 409-429.
[29] Nguyen, T. T., & Reddi, V. J. (2021). Deep reinforcement learning for cyber security. IEEE Transactions on Neural Networks and Learning Systems, 34(8), 3779-3795.
[30] Zhang, Y., Liu, J., Zhou, S., Hou, D., Zhong, X., & Lu, C. (2022). Improved deep recurrent q-network of POMDPs for automated penetration testing. Applied Sciences, 12(20), 10339.
[31] Li, Y., Dai, H., & Yan, J. (2024, June). Knowledge-informed auto-penetration testing based on reinforcement learning with reward machine. In 2024 International Joint Conference on Neural Networks (IJCNN) (pp. 1-9). IEEE.
[32] Zhou, T. Y., Zang, Y. C., Zhu, J. H., & Wang, Q. X. (2019). NIG-AP: A new method for automated penetration testing. Frontiers of Information Technology & Electronic Engineering, 20(9), 1277-1288.
[33] Chen, Z., Kang, F., Xiong, X., & Shu, H. (2024). A survey on penetration path planning in automated penetration testing. Applied Sciences, 14(18), 8355.
[34] Chen, Y., He, J., Fang, W., Yang, S., Chen, J., Li, T., & Lan, X. (2025). Automatic penetration testing model based on reinforcement learning for complex network environments. The Journal of Supercomputing, 81(11), 1-27.
[35] Yang Chen,Junjiang He,Wenbo Fang,Shenwen Yang,Jiangchuan Chen,Tao Li & Xiaolong Lan. (2025). Automatic penetration testing model based on reinforcement learning for complex network environments. The Journal of Supercomputing,81(11),1169-1169.
[36] Fucai Luo,Tingfa Xu,Jianan Li & Fengxiang Xu. (2025). MDP-AD: A Markov decision process-based adaptive framework for real-time detection of evolving and unknown network attacks. Alexandria Engineering Journal,126,480-490.
[37] Yutao Hu,Yongxin Feng,Yuntao Zhao & Xuedong Mao. (2025). IoT-ONDDQN: A detection model based on deep reinforcement learning for IoT data security. Computer Communications,241,108263-108263.
[38] Qiuhan Dong,Jinjin Tang,Wen Long Shang,Xinyun Shao & Mikela Chatzimichailidou. (2025). Intelligent scheduling of urban rail transit loop line trains: A study on coordinated optimization of timetables and rolling stock circulation plans based on DQN deep reinforcement learning. Computers & Industrial Engineering,207,111297-111297.