Hindawi
Security and Communication Networks
Volume 2023, Article ID 5834434, 15 pages
https://doi.org/10.1155/2023/5834434

Research Article

DQfD-AIPT: An Intelligent Penetration Testing Framework Incorporating Expert Demonstration Data

Yongjie Wang,[1,2] Yang Li,[1,2] Xinli Xiong,[1,2] Jingye Zhang,[1,2] Qian Yao,[1,2] and Chuanxin Shen[1,2]

[1]*College of Electronic Engineering, National University of Defense Technology, Hefei 230037, China*
[2]*Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation, Hefei 230037, China*

Correspondence should be addressed to Yang Li; ly_20d@163.com

Received 9 August 2022; Accepted 7 October 2022; Published 4 May 2023

Academic Editor: Muhammad Faisal Amjad

The application of reinforcement learning (RL) methods of artificial intelligence for penetration testing (PT) provides a solution to the current problems of high labour costs and high reliance on expert knowledge for manual PT. In order to improve the efficiency of RL algorithms for PT, existing research has considered bringing in the knowledge of PT experts and combining it with the use of imitative learning methods to guide the agent in its decision-making. However, the disadvantage of using imitation learning is also obvious; that is, the performance of the strategies learned by the agent hardly exceeds the demonstrated behaviour of the expert and it can also cause expert knowledge overfitting. At the same time, the expert knowledge in the currently proposed method is poorly interpretable and highly scenario-dependent. The expert knowledge used in these methods is not universal. To address these issues, we propose an intelligent PT framework named DQfD-AIPT. The framework encompasses the process of collecting and using expert knowledge and provides a rational definition of the structure of expert knowledge. To solve the overfitting problem, we perform PT path planning based on the deep Q-learning from demonstrations (DQfD) algorithm. DQfD combines the benefits of RL and imitation learning to effectively improve the PT strategy and performance of agents while avoiding overfitting. Finally, we conducted experiments in a simulated network scenario containing honeypots. The experimental results proved the effectiveness of expert knowledge incorporation. In addition, the DQfD algorithm can improve the efficiency of penetration testing more effectively than that by the classical deep reinforcement learning (DRL) method and can obtain a higher cumulative reward. Not only that, due to the incorporation of expert knowledge, in scenarios with honeypots, the DQfD method can effectively reduce the probability of interacting with honeypots compared to the classical DRL method.

1. Introduction

With the development of the Internet, the network environment is increasingly complex, and cyber security threats that we face are increasing day by day. Globally, protecting modern systems and infrastructure is becoming a challenge in the field of computer security. The traditional approach to system security assessment takes a defender's perspective by solidifying and enhancing system security against attackers [1]. Penetration testing is used as a positive method to attack and test a target system against authorised networks from the attacker's point of view. We can conduct vulnerability detection and security assessment through potential threat paths [2]. However, with the increase in network size and system complexity, the number of hosts in the network, and the complexity of configuration information, the efficiency of performing penetration testing will be affected by the AoI (age of information) [3, 4]. Performing PT manually involves a lot of repetitive actions and procedures [5]. As a result, automated and intelligent penetration testing was born out of this need.

Early research included automated penetration testing tools and related theoretical studies. Automated penetration testing tools integrate modules for scanning, penetration attacks, and payload selection, such as Metasploit [6]. However, human intervention is still required for the critical

target identification and load selection in this tool. In terms of theoretical research, attack trees, attack graphs, and planning domain programming languages (PDDLs) are representative. Methods such as attack graphs plan attack paths through formal representation of the target network configuration information and state transfer analysis [6, 7]. However, all these methods require in-depth knowledge of the target network's information in advance and cannot model the uncertainty in the penetration testing process. Recent advances in artificial intelligence have provided a new approach to the current research on automated and intelligent penetration testing. In particular, RL methods are proving to be a general and effective approach. RL can be used to solve the problem of optimal performance of an agent in a given environment. A model-free approach to RL, for example, allows an agent to learn a strategy by interacting with the environment, with little reliance on prior knowledge of the environment. The method is analogous to a human player interacting with a game in order to complete the game objectives and learn relevant solution strategies [8].

The Markov decision process (MDP) is a paradigm of RL. Schwartz and Hanna [9] and Zhou et al. [10] trained the agent for path planning by formalising PT as MDP and using the DRL algorithm in a constructed penetration test simulation environment. The problem is that the training process takes a long time to converge and that the simulation environment is poorly simulated. Zennaro et al. [11] combine RL and imitation learning approaches and classify PT problems according to scenarios. They provide a priori knowledge to the agent for a specific network scenario structure, which better guides the agent to explore their problem space and thus obtain a better solution. On the downside, its expert knowledge is limited by the constructed penetration test scenarios and is less interpretable and generalisable. Chen [12] proposed an intelligent PT framework named GAIL-PT. GAIL-PT collects expert knowledge via Metasploit and uses GAIL (generative adversarial imitation learning) in imitation learning for path planning. However, the complexity of the A3C-GAIL and DPPO-GAIL algorithms used in the experiments is still high. The above research takes full account of the characteristics of the penetration testing problem and uses imitative learning methods for intelligent penetration testing while incorporating expert knowledge. The use of expert knowledge with decision aids as inputs can go some way to improving the PT strategy of the agent, allowing it to be trained in a direction close to the behaviour of the expert [13]. However, the aim of imitation learning methods for training models is to fit the trajectory distribution of model-generated strategies to the trajectory distribution of the input. Therefore, using imitation learning in combination with RL makes it difficult to make policy enhancements to that part of the environment that has not been explored. In conclusion, the following challenges need to be stressed:

(i) Challenge 1: the penetration testing expert knowledge provided to the agent is poorly interpretable, usually dependent on specific network scenarios, and not universally applicable.

(ii) Challenge 2: the use of imitation learning is tending to produce an overfitting of expert knowledge, making it difficult to balance exploration and exploitation of the environment. The higher complexity of the algorithms used for intelligent penetration testing leads to slower convergence and lower efficiency.

To address these challenges, we first propose an intelligent PT framework called DQfD-AIPT that incorporates expert knowledge. DQfD-AIPT contains mainly the collection and exploitation of expert knowledge and the training of agents. At the same time, we have rationalised expert knowledge. Second, we use the DQfD algorithm based on the reinforcement learning with expert demonstrations (RLED) framework, combining both supervised and unsupervised methods to construct the loss function. The algorithm also uses prioritized experience replay (PER) for experience sampling to balance expert data and interaction data to prevent overfitting. Ultimately, experiments were conducted in a simulated network scenario containing honeypots to verify the effectiveness of expert knowledge incorporation and to test the performance of the DQfD algorithm. The experiments show that the DQfD algorithm has better penetration testing performance than the classical DRL method. For the experimental platform, we selected the CyberBattleSim (CBS) platform developed by Microsoft, which features high simulation and support for RL algorithms and is currently a more well accepted intelligent PT simulation platform.

The main contributions of this paper are as follows:

(i) To address the problems of poor interpretability of expert knowledge and dependence on specific scenarios, we propose an intelligent PT framework named DQfD-AIPT that incorporates expert knowledge. The construction of an expert knowledge base is carried out through two methods: the transformation of abstract expert knowledge and the collection of PT traces in different network scenarios. At the same time, we also define the form and structure of expert knowledge.

(ii) To address the problem of overfitting of expert behaviour due to imitation learning, we use the DQfD algorithm incorporating expert demonstration data, together with a PER mechanism for sampling of expert data and interaction data. With the guidance of the expert demonstration data and interaction data, the efficiency and overall performance of the training process of the agent are effectively improved.

The organization of this paper is as follows: In Section 2, we provide an overview of research advances in PT, intelligent PT, and use of expert knowledge. In Section 3, we explain and outline the RL covered in this paper. In Section 4, we describe the method that we used and the specific implementation details of the method. In Section 5, we describe the procedure and hyperparameter settings of the experiments, while the results are analysed and evaluated.

Finally, in Section 6, we summarise our work and further articulate future research directions.

2. Related Work

In this section, we mainly introduce the basic concepts of PT, the research progress of intelligent PT, and the importance of expert knowledge in the PT process. In the end, we conclude with a summary of the problems in the current study.

2.1. PT and Its Automation. PT is a method of simulating real attacks with the aim of assessing the security of computer systems and networks. PT assesses information security from the attacker's perspective. Through PT of companies, organisations, or departments, we understand their information security policies and network vulnerabilities and give possible solutions and remedies to improve network security [14]. As network equipment and defence detection systems continue to upgrade, the complexity of performing the PT process has increased dramatically. The entire testing process involves skilled cyber security experts generating attack plans to discover and exploit vulnerabilities in networks and applications. A team of highly experienced testers is therefore essential, who must control all tasks manually. In addition to this, there are a large number of repetitive actions and deterministic steps in the PT process, which will lead to the problem of the high time cost of conducting PT manually. In summary, PT currently appears to be a costly means of assessing the vulnerability of network systems [15, 16].

To address the high cost and reliance on manual PT, methods and tools to implement automated PT have been proposed. In terms of theoretical research, early studies such as attack trees, attack graphs, and PDDL are representative. These methods plan attack paths through formal representations of target network configuration information and state transfer analysis [3, 4]. On the one hand, these abovementioned methods require full knowledge of the network topology and the configuration of each machine, which is unrealistic from the attacker's point of view. On the other hand, these methods focus on a regular representation of known information and then find the path of attack by means of planning. The uncertainty of a real PT process is not well modeled. That is, the uncertainty of system knowledge must be obtained using remote tools before a planned attack can be executed.

In terms of the development of automated PT tools, mature automated PT tools include APT2 PT suite, Autosploit PT tool, and Awesome-Hacking tools [2]. These PT tools have significantly improved the efficiency of PT and simplified the process of conducting PT manually. However, there is still the problem of not being able to intelligently select the attack payload and of targeting only a single host. There is still a need to base the correct choice of attack methods and means on the decisions of PT experts. The use of intelligent planning techniques to improve the automation of attack path discovery is still the key to achieving automated PT [17].

2.2. Intelligent PT Using RL Methods. With the development of algorithms in the field of artificial intelligence, there have been new advances in the study of automated PT. Artificial intelligence-driven PT methods are able to intelligently select attack targets and attack payloads based on the current state of the target network. The RL approach learns how to map the current state to an action and provides an idea how to do so. The agent learns the PT strategy by interacting with the environment composed of the target network and based on the feedback from the interaction. The process by which we build a simulated environment for PT to train an agent is similar to how a player interacts with a game to discover its solution [9].

A precondition for applying RL for intelligent PT is the need to formalise the PT process into the RL paradigm. Sarraute et al. [18], Schwartz et al. [19], Hu et al. [20], and Zennaro [11] et al. formalised the penetration testing process as a partially observable Markov decision process (POMDP). They incorporated the attacker's observations of the environment into the attack process. However, as the size of the network scenario expands, the computational complexity increases and is still not applicable to large-scale networks. Durkota and Lisý [21] proposed the model penetration test as an MDP, in which the action space consists of specific vulnerabilities and the state space consists of the results of attack actions. The goal of the whole model is to minimize the expected loss value. Hoffmann [22], based on Durkota, ignores the structure of the target network and relies instead on expressing the uncertainty of PT in the form of possible action outcomes. This is essentially a model-free approach [9] that requires minimal prior knowledge of the environment. However, POMDP is more realistic in most cases. However, considering the computational complexity and the efficiency of the reinforcement learning algorithm, the MDP model is still a better scheme to balance the computational efficiency and modeling rationality.

In recent years, a variety of RL algorithms have been used extensively in addressing intelligent PT. Schwartz and Hanna [9] constructed the network attack simulator (Nasim) and used known network configurations as states, available scans and exploits as actions, and used table-based Q-learning methods and neural network-based DQN methods to achieve intelligent discovery of attack paths. Zhou et al. [10] combined various improvements with the DQN algorithm and proposed the NDSPI-DQN algorithm to optimise PT path discovery. The algorithm effectively reduces the action space of the agent and is experimentally validated based on Nasim. Zhang [2] introduced the multidomain action selection module on the basis of intelligent PT. This module can effectively identify the actions that can be used according to a specific state, reducing unnecessary exploration by an agent. Finally, this method combined with the deep deterministic policy gradient (DDPG) algorithm is verified in a simulated environment.

2.3. Use of Expert Knowledge. Expert guidance often plays a key role in solving real-world problems. As a method highly dependent on expert knowledge, reference to

experienced expert knowledge is often helpful to exploit the vulnerability of the target system, so as to achieve the target at a lower cost. From the perspective of research status, the current research is mainly focused on solving the problems of state space explosion, action space explosion, and sparse reward caused by penetration testing using the reinforcement learning algorithm. Most of them focus on the algorithm itself, often ignoring the characteristics of expert decision-making in the penetration test process and the analysis of specific network scene structures.

Zennaro et al. [11] simplified the penetration testing problem with different structures in the form of a capture flag challenge and demonstrated how the performance of an agent can be improved by relying on different forms of prior knowledge provided to the agent. The experiments show that by incorporating prior knowledge, the agent can better explore the space of their problems and thus effectively obtain solutions. However, the CTF scenarios constructed for the experiments were only simplified versions and were not experimented on relatively complex scenarios. Chen [12] first proposed a generic intelligent PT framework based on GAIL. GAIL-PT addresses the problem of high labour costs due to the intervention of security experts and high-dimensional discrete action spaces. The study used a variety of algorithms for experiments, but the results showed that the complexity of the A3C-GAIL and DPPO-GAIL algorithms, which combine GAIL, is still relatively high.

The main idea of imitation learning is to match the behavioural strategies of an agent with the behaviour of an expert by means of training. Imitation learning can be divided into behavioural cloning [23], inverse reinforcement learning [24], and generative adversarial imitation learning [25]. However, imitation learning tends to focus on imitating the behaviour and trajectories of experts, making it difficult to enhance and contribute to the performance of the agent in the environment when combined with RL methods. This method is essentially an exploration and exploitation of the environment without enhancement for the strategy in the application of RL methods. Recently, demonstration data have been shown to help solve difficult exploration problems in RL. Subsequently, a framework known as reinforcement learning with expert demonstration (RLED) was proposed. This framework is suitable for scenarios where rewards are provided by the environment. Todd Hester et al.of the Google DeepMind team [26] propose the DQfD algorithm based on the RLED framework. The method is pretrained on presentation data while combining the features of supervised and unsupervised learning to construct a loss function and using PER in order to achieve a balanced amount of demonstration data in the training data. By training deep neural networks in this way, the results show that DQfD outperforms imitation learning, which only imitates expert trajectories, as well as classical deep Q networks in terms of average overall performance.

2.4. Brief Summary. We have summarised the current progress in intelligent penetration testing research in the previous section. Traditional penetration testing has high reliance on expert knowledge and high labour costs. In the process of changing from manual to intelligent PT, the cost of manual time is effectively reduced. This goes some way to solving one of the major dilemmas of current manual PT. However, as the complexity of the target system increases, the performance of penetration testing using classical RL methods also encounters bottlenecks. The problem is that real-world PT relies on expert experience and the proficiency of the penetration tester. Knowledge reasoning between successive states during real PT is missing in the training of the agent, but in the course of a real penetration test, this is quite important. Considering the integration of expert knowledge into intelligent PT can effectively solve these problems.

In the exploration of incorporating expert knowledge in PT, previous studies have mainly used imitative learning methods. These studies have been conducted by processing collected expert knowledge and combining it with imitative learning methods for PT path planning. On the one hand, the expert knowledge collected is usually highly relevant to the target network scenario and not universally applicable. The interpretability of this expert knowledge is relatively poor. On the other hand, the use of a combination of imitative and RL was effective in guiding the agent to fit the expert knowledge trajectory to some extent. However, imitative learning is more concerned with imitating the behaviour of the expert than the strategies of the testing expert. This will lead to problems of expert knowledge overfitting. As a result, agents trained using imitation learning often struggle to outperform experts. In conclusion, the current research in the field of intelligent penetration testing incorporating expert knowledge can be further enhanced with regard to the interpretability of expert knowledge and the RL methods used.

3. Preliminaries

3.1. Classical RL Method and Its Improvements

3.1.1. Q-Learning. Q-learning is a value-based RL algorithm. Q is $Q(s, a)$, which is the expected gain from taking an action $a\,(a \in A)$ in a state $s\,(s \in S)$ at a given time. The main idea of the algorithm is to construct a Q table of states and actions to store Q values and then select the action that yields the greatest benefit based on the Q value. Q-learning uses temporal difference (TD) to update Q values, with the updated formula shown in the following equation:

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha\left[r_t + \gamma \max_{a_{t+1}}(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right], \tag{1}$$

where α is the learning rate, γ is the discount factor, a_t and s_t are the action and state at moment t, respectively, s_{t+1} is the next state after performing action a_t, a_{t+1} is the possible action to be performed in the state s_{t+1}, and r_t is the immediate reward obtained.

3.1.2. Deep Q-Learning. Q-learning takes a tabular approach to storing Q values. Therefore, when facing the RL

assignment with a high-dimensional state space and action space, the limited space of the table cannot store all states and actions, which limits the performance of the algorithm [27]. Algorithms that combine the advantages of deep learning give a better solution to this problem. Minh [28] proposed the deep Q-network (DQN), which is an extension of Q-Learning. The algorithm replaces the Q value table in Q-learning with a neural network. This transforms the original problem of convergence of action value function solving into a function fitting problem for neural networks.

During the exploration and exploitation of the environment, the transition data are stored in a replay buffer. DQN uses randomly sampled data from the replay buffer to train the neural network to approximate the Q function. This method breaks the correlation between the training data and makes the training process stable. The DQN algorithm uses the square of the error between the target Q value and the estimated Q value as the loss function when updating the parameters of the neural network, where the target Q value y_i at the i iteration is calculated as follows:

$$y_i = E_{s_{i+1}\sim\varepsilon}\left[r_i + \gamma \max_{a_{i+1}} Q'\left(s_{i+1}, a_{i+1} \mid \theta_i'\right) | s_i, a_i \right], \tag{2}$$

where $\theta_i^{'}$ are the parameters of the target Q network. The goal of strategy learning is to update the parameters of the strategy Q network by the mean square error between the target Q value and the current Q value, where the loss function of the algorithm at the i iteration is calculated as follows:

$$L_i\left(\theta_i\right) = E_{s_i, a_i \sim \rho(\cdot)}\left[\left(y_i - Q\left(s_i, a_i \mid \theta_i\right)\right)^2 \right], \tag{3}$$

where $\rho(s_i, a_i)$ is the probability distribution of s_i and a_i. θ_i are the parameters of the strategy network. The parameters of the strategy network are assigned to the target network in fixed steps of intervals. When optimizing the loss function, the parameters of the target network $\theta_i^{'}$ are not updated.

3.1.3. Prioritized Experience Replay. Prioritized experience replay is a technique for prioritizing experiences, whereby important state transition experiences are replayed more frequently. This method is effective because some transition data contain more information that is worth learning. Giving these transitions more opportunities to be played back helps accelerate the overall learning process. The core idea of PER is to measure the importance of different transition data through the TD error δ. The larger the error of the sample, the larger the value of the sample. The sampling probability of the state transition i is calculated as follows:

$$P(i) = \frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}}, \tag{4}$$

where p_i refers to the priority of the state transition i, denoted as $p_i = |\delta_i| + \in$, and $\in$ is a positive number used for numerical stability so that $p_i \rangle 0$. ∂ is an exponential hyperparameter representing the degree of the impact of the TD error on the sample.

It is worth noting that the purpose of using experience replay is to eliminate sample correlation, but the use of prioritized sampling certainly forgoes random sampling. Therefore, it is also necessary to reduce the training weights of the high-priority state transition data. The PER method uses importance sampling weights to correct for deviations in the state transition i. The weights are calculated as follows:

$$\omega_i = [NP(i)]^{-\beta}, \tag{5}$$

where N refers to the capacity size of the replay buffer and β is the annealing hyperparameter of the training process. To implement the above method efficiently, we store the priorities in an efficient query line tree data structure and sample the range of line segments during the training process. We call this efficient query data structure a sum tree.

3.2. RL Formalisation for PT. An RL agent must be able to perceive the state of the environment, have one or more goals related to the state of the environment, and then take action and influence the state of the environment. In order to implement intelligent PT in conjunction with an RL method, we begin by modeling the penetration testing process as an RL paradigm. MDP is a theoretical framework for achieving goals through interactive learning. It is the classic formal expression of sequential decision-making. The actions of an agent in an MDP affect not only the current immediate reward but also the subsequent state and the future benefit. Thus, the MDP is a mathematically idealised form of the reinforcement learning problem. The interaction process between the agent and the environment in the MDP is shown in Figure 1.

The machines that learn and implement decisions in the MDP are called agents. Everything that interacts with it outside of the agent is referred to as the environment. Taking the penetration testing process as an example, if the target network is considered as a state variable environment, the feedback from the environment to the actions of the agent is considered as a reward. The whole penetration testing process can then be represented in the form of a 4-tuple$\langle S, A, R, T\rangle$, where S represents the state space, A represents the action space, R represents the reward function, and T represents the transfer function. Detailed definitions for specific penetration testing questions are given in Section 4.3.

4. Methods

In this section, we first present an intelligent PT framework incorporating expert knowledge and explain the details and processes involved in this framework. We then detail the collection and use of expert knowledge and present the RL algorithm that we use that incorporates demonstration data.

4.1. DQfD-AIPT Framework. Manual PT relies on the experience and knowledge of experts. Expert knowledge and the way of making decisions are also of great importance for intelligent PT. By analysing the PT process and summarising

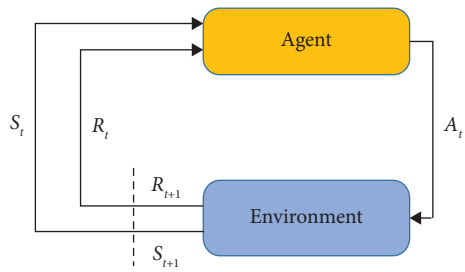

Figure 1: The agent interacts with the environment.

the characteristics of expert knowledge and RL, we propose DQfD-AIPT for intelligent PT that incorporates expert knowledge as shown in Figure 2. DQfD-AIPT consists of three main phases: expert knowledge collection, input and use of demonstration data, and interaction and training of the agent. Our proposed framework is suitable for intelligent PT in simulated network scenarios and is characterised by simplicity of use and generality. In the following, we will explain the specifics of each phase of the framework.

4.1.1. Stage 1: Collection of Expert Knowledge. The first step in combining expert knowledge for intelligent PT is the collection and acquisition of expert knowledge. The acquisition of expert knowledge is an abstract and difficult matter. The difficulty lies in the fact that the generation and design of expert knowledge are often based on the experience and rules of experts. As the process of PT is usually strongly correlated with the structure and vulnerability distribution of the target network scenario, the actions taken by penetration testers facing a specific target network are somewhat unpredictable. Considering the interpretability and validity of expert knowledge, we propose two ways of collecting expert knowledge:

(i) Method 1: As shown in Figure 3, first, we transform the abstract experience of the penetration tester into an executable action that can interact with the simulated environment. This executable action is usually mapped to a specific environmental state. For example, a penetration tester decides to take a certain penetration exploit action based on the current state of the target network. We can abstract this expert knowledge and represent it in the form of a state-action pair. Afterwards, the penetration test simulation environment executes the action and processes the result of the action and gives feedback. Finally, the reward value R resulting from the execution of action A is integrated into the state S together with the new state S'. We obtain a complete set of expert transition data that can be used by the training of the agent and stored in an expert knowledge base.

(ii) Method 2: We collect valid paths and traces of agents completing PT objectives in multiple different simulated network scenarios (scenarios that are within a fixed order of magnitude due to the uniformity of expert knowledge). These trajectories consist of multiple transition data, and the transition data are obtained from tests of the agent against different network scenarios. However, due to the structural specificity of the expert data that we designed, real-world common open ports and services, for example, have specific bitmasks in the transition data. These predefined bitmasks often override the configuration of our simulated network environment. For example, the agent collects expert data in multiple network scenarios of size less than N. We can apply this expert knowledge to the training of network scenarios of a size less than N. In addition to this, we can take valid transition data (reward values > 0 for action execution) and store them in an expert knowledge database.

4.1.2. Stage 2: Input of Expert Knowledge. The transitional data collected in the expert knowledge base can be used as demonstration data to ensure that agents can learn the PT strategy of experts through pretraining. The demonstration data input to the agent conform to the standard transition data form of reinforcement learning algorithm and are used for training and processing. The detailed representation and structure of transition data are shown in Figure 3.

The transition data consist of a four-tuple: $\tau(S, A, R, S')$, where S is a valid observation of the current target network state by the agent and also serves as the state input to the neural network in the RL algorithm, A is the vector of actions performed, consisting of the number of the agents' actions, R is the reward value for executing action A under the state S, and S' is the new state to which the transition is made after the execution of action A under the state S. Each transition in the expert knowledge base has the same structure and can be applied to the training of the agent as demonstration data. The observed state contains the agent's perception of the target network environment, which contains statistically tractable information that has an impact on the action decisions of the PT process. This information includes, among other things, the number of hosts the agent has discovered, the number of hosts it controls, the number of open services it has scanned, the number of connection credentials it has obtained, and the ports it has discovered.

4.1.3. Stage 3: Interaction and Training of the Agent. As the agent interacts with the penetration test simulation environment, the agent's actions will change the state of the environment, while the environment will give feedback to the agent on rewards and penalties. The agent adjusts the PT strategy and actions based on the rewards. At this point, the demonstration data extracted from the expert knowledge base serve to assist the agent in making decisions and to influence the agent's tendency to perform actions. The better transition data generated by the agent as it interacts with the environment are also recorded and stored in the expert knowledge base. In this way, the expert knowledge base is continuously expanded with valid transition data. More

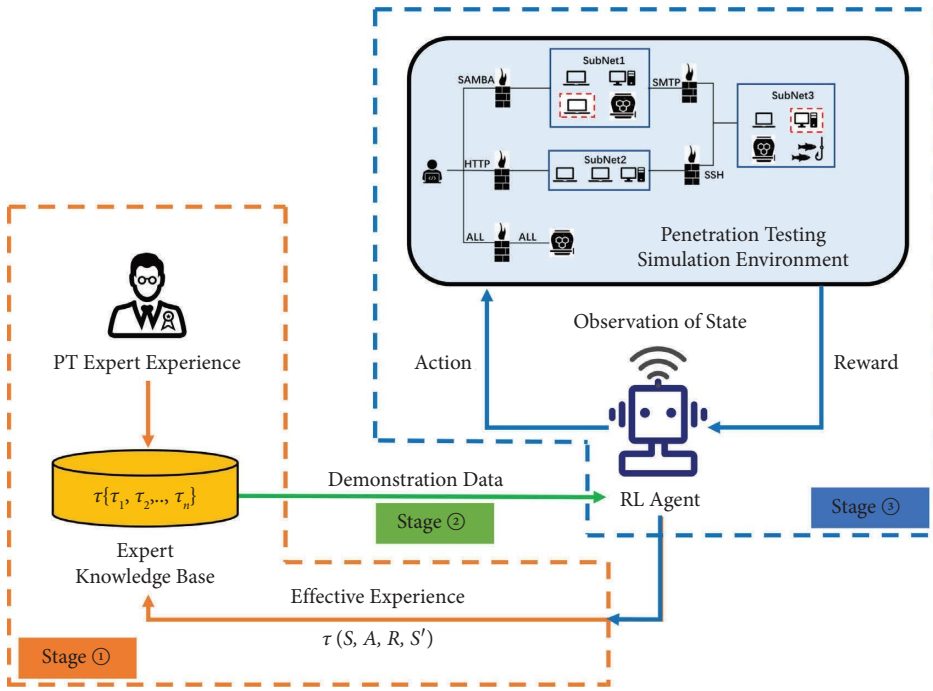

Figure 2: Framework structure of DQfD-AIPT.

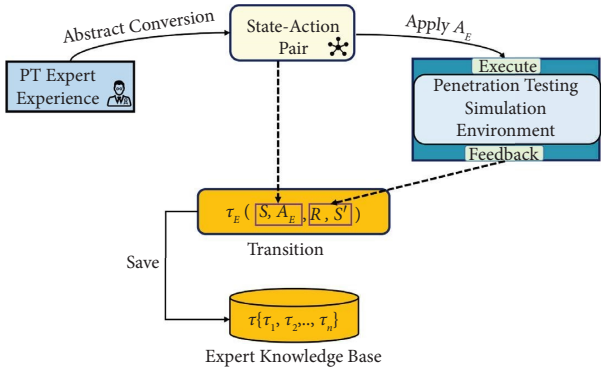

Figure 3: The process of converting abstract expert knowledge into transition data.

comprehensive and effective guidance is provided for the training of the agent.

4.2. DQfD Training. The previous section described the basic process of implementing our DQfD-AIPT framework incorporating expert knowledge. In particular, we highlighted the process of collecting expert knowledge data. This section will describe the DQfD RL algorithm and the specific implementation details of how the algorithm combined with expert knowledge data can guide the agent in PT.

Based on the algorithm structure of DQN and combined with the framework of DQfD, we implement the DQfD algorithm, whose algorithm structure can be expressed as shown in Figure 4. First, based on the algorithmic structure of the DQN, we build a policy network and a target network with the same structure. Each network consists of a multilayer structured neural network: an input layer, three fully connected hidden layers, and an output layer. At the same time, we implement PER through the tree storage structure of the sum tree. PER draws those samples that are more valuable at higher frequency than random samples, and PER takes into account the importance of the different state transition data by means of the TD error. This approach is used to balance the proportion of demonstration data and interaction data contained in the small batch of transition data sampled. On this basis, we combine the demonstration data from the expert knowledge base to improve algorithm performance and learning efficiency.

The detailed process of the DQfD algorithm can be described as follows.

4.2.1. Pretraining Stage. State transition data from the constructed expert knowledge base are prepositioned in the demonstration data area of the sum tree. In particular, it is important to emphasise that the size of the demonstration data area and the data filled by the sum tree are fixed. Throughout the training process, the data in the demonstration area are not overwritten as new state transition data are added to the sum tree. After the expert knowledge preset was completed, the policy network was pretrained by sampling batch-sized state transition data from the demonstration data areas several times. The pretraining process updates the parameters of the Q network using a $J(Q)$ loss combining the three losses, where the $J(Q)$ error is calculated as follows:

$$J(Q)_s = J_Q(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q), \quad (6)$$

where $J(Q)_s$ is the joint loss containing the supervised loss, λ_1, λ_2, and λ_3 represent the constants, respectively, $J_Q(Q)$ is the loss of DQN, $J_n(Q)$ is the N-step return loss, $J_E(Q)$ is

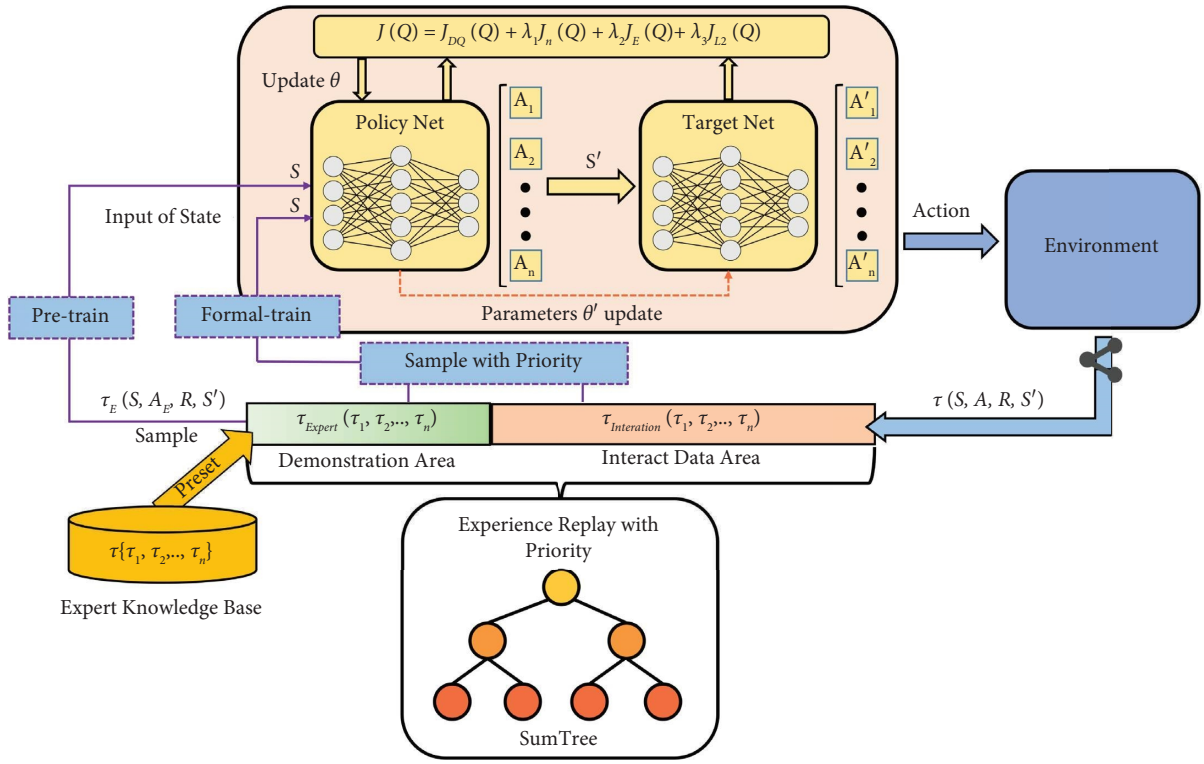

Figure 4: DQfD algorithm training process.

a supervised loss, and $J_{L2}(Q)$ is a regularisation term applied to the neural network to alleviate overfitting to the presentation data and to prevent the strategy from overfitting to a small fraction of the experience in the expert data. A detailed explanation of the $J_n(Q)$ and $J_E(Q)$ is given below.

For the $J_n(Q)$ loss, the agent updates its Q-network with a mixture target of 1-step and n-step return. The incorporation of the N-step loss can help propagate the value of the expert data to an earlier state, greatly enhancing learning from the limited demo dataset. It also ensures that the pretrained learned neural network value function estimates satisfy the Bellman equation. The calculation of $J_n(Q)$ can be expressed as follows:

$$r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \max_\alpha \gamma^n Q(s_{t+n}, a). \quad (7)$$

$J_E(Q)$ is a supervised loss. The incorporation of the supervised loss is the key to the pretraining process. It can be expressed as follows:

$$J_E(Q) = \max_{a \in A} [Q(s,a) + l(a_E, a)] - Q(s, a_E), \quad (8)$$

where a_E is the action corresponding to the expert demonstration data in the state S. $l(a_E, a)$ is a margin function, which is a measure of how well the currently executed action matches the action demonstrated by the expert and can be expressed as follows:

$$l(a_E, a) = \begin{cases} 0, & a_E = a, \\ k, & a_E \neq a. \end{cases} \quad (9)$$

In this way, the value of any action that differs from the expert action a_E is less than the value of the action a_E. With the supervised loss, the values of actions outside the range of the demonstration data also become reasonable values, resulting in a value-driven ε-greedy strategy that effectively imitates the expert's actions. Pretraining is a good starting point for learning the task. Once the agent begins to interact with the task, it continues to learn by sampling from its own generated and demonstrated data.

4.2.2. Training Stage. After the pretraining phase, we get an agent with expert experience. However, the agent does not interact with the environment throughout the pretraining phase. During the formal training phase, the agent first interacts with the environment to generate transition data, and each transition data is stored in the interaction data area of the sum tree structure. In addition, to avoid overfitting of the expert transition data early in the training process, the interaction data area of the sum tree needs to be filled up by the interaction of the agent with the environment before the formal learning begins. After the maximum storage capacity is reached, the agent-generated data will continuously cover the interaction data area of the sum tree structure. The flow of the PER algorithm is presented in Algorithm 1.

In the pretraining phase, only the expert transition data are extracted from the demonstration data areas for training. In the formal training phase, the transition data from both the demonstration data region and the interaction data area could be extracted from the sum tree according to the PER method. The difference is that the supervised loss $J_E(Q)$ is

Input: k: batch size, N: capacity of the sum tree, and n: the amount of transition currently stored by the sum tree, initialised the sum tree structure
Output: Updated the sum tree structure after sampling the transitions
(1) **if** $n < N$ **then**
(2) push the transition $\tau(S, A, R, S')$ into the sum tree with maximal priority
(3) **end if**
(4) Start sampling batch-size transitions from the sum tree
(5) Calculate $\sigma \leftarrow$ SumTreeMaxPriority/k
(6) **for** steps $t \in \{1, 2, \ldots k\}$ **do**
(7) Sample k transition data with priority from the sum tree
(8) $a \leftarrow t^* \sigma$, $b \leftarrow (t+1)^* \sigma$,
(9) $v \leftarrow$ generate a random number between a and b
(10) Transition τ_t and corresponding priority are obtained according to a random number v
(11) Compute importance sampling weight for each transition τ_t
(12) **end for**
(13) Train this batch size of transition and compute the TD error according to the weight
(14) Update transition priority according to the TD error

Algorithm 1: Implements PER with the sum tree structure.

removed from the calculation of the joint loss $J(Q)_s$, which indicates that $\lambda_2 = 0$.

In addition to this, the network update process is more computationally expensive due to the forward propagation process compared to the forward propagation process. The purpose of this form is to ensure that the replay buffer is closer to the state distribution of the current policy and to prevent network overfitting. Therefore, the update frequency of the target network in the pretraining phase is measured in steps, and the step setting interval should not be too small. In the formal training stage, the update frequency of the target network is in episodes. In summary, the DQfD algorithm combined with expert knowledge is presented in Algorithm 2.

4.3. RL Settings for PT. In this paper, we model the PT process as an MDP. We use the RL agent as an attacker who penetrates the target system. The target system constitutes the environment in which the agent interacts with each other. We take the formalisation of MDP as a basis and consider the characteristics of the intelligent PT simulation environment used for the experiments. We give the relevant settings for the necessary elements required for RL. The relevant elements in the PT formalisation into an MDP can be represented separately as follows:

(i) State space: A state space is a finite set of states with a nonfixed structure. In PT problems, the state space covers the range of changeable states of the target network. In this paper, we take the awareness of the target network environment by an agent through observation as the state. The representation of the specific states is shown in Figure 5.

(ii) Action space: The action space contains all the executable actions of the agent and does not change with the current state of the environment. This means that the output dimension of the neural network is always fixed during the state-to-action mapping process. In this paper, there are three main types of actions for an agent:

① Local exploit: the local exploit action is the process of exploiting the local resources of a target host after taking control of that host. The outcome of this action exploitation is privilege elevation, credential information leakage, suspicious link leakage, etc.
② Remote exploit: the remote exploit uses the current controlled host as a springboard to execute malicious commands by submitting them in the local browser. The outcome of the exploit is to gain control of the target host, leaking a suspicious link, etc.
③ Connect: the connect action acts on the discovered hosts and connects to the target host by means of the host credential information acquired during the lateral movement. The action outcome is to control the target host.

(iii) Reward: the reward function is feedback from the environment for the action taken by the agent, and the calculation of the reward during the penetration test can be expressed as follows:

$$R = \text{Eval}\left(\text{Outcome}_A^S\right) - \text{Cost}(\text{Action}). \tag{10}$$

In the equation, $\text{Eval}(\text{Outcome}_A^S)$ is an evaluation of the outcome of the execution of an action by the agent. Cost (Action) is the cost of performing the action. The classification of the exploit outcomes of the actions and the corresponding values are shown in Table 1.

(iv) Transfer function: The transfer function is a description of the probability of the environment to make a state transfer under certain conditions. For the model-free approach, learning is performed from the experience generated during the interaction, as the agent cannot directly rate the merit

Input: D^{replay}: the experience replay area built by the sum tree, D^{demo}: expert demonstration data area in D^{replay}, $D^{interact}$: interactive data area in D^{replay}, θ: weights for the policy network (randomly generated), θ': weights for the target network (randomly generated), f_p: update target network frequency of pretraining, f_f: update target network frequency of formal training, k: batch size, j: number of pretraining gradient updates, E: episode number of training, and S: max steps per episode
Output: An agent trained with expert knowledge

(1) Push expert transition data into D^{demo} and initialize their priority
(2) **for** steps $t \in \{1, 2, \ldots j\}$ **do**
(3) Sample a batch size of k transitions from D^{demo} with prioritization
(4) Calculate loss $J(Q)_s$ using the target network
(5) Perform a gradient descent step to update the weights for the policy network θ
(6) **if** $t \bmod f_p = 0$ **then** $\theta' \leftarrow \theta$ **end if**
(7) **end for**
(8) **for** episode $u \in \{1, 2, \ldots E\}$ **do**
(9) **for** step $v \in \{1, 2, \ldots S\}$ **do**
(10) Sample action A from the behaviour policy
(11) The environment performs A and gives back R(reward), and the agent observes (S, A, R, S')
(12) Push the transition $\tau(S, A, R, S')$ into $D^{interact}$, overwriting oldest interaction transition if over capacity of $D^{interact}$
(13) Sample a batch size of k transitions from D^{replay} with prioritization
(14) Calculate loss $J(Q)$ using the target network
(15) Perform a gradient descent step to update the weights for the policy network θ
(16) $S' \leftarrow S$, the state transitions from S to S'
(17) **end for**
(18) **if** $u \bmod f_f = 0$ **then** $\theta' \leftarrow \theta$ **end if**
(19) **end for**

Algorithm 2: DQfD.

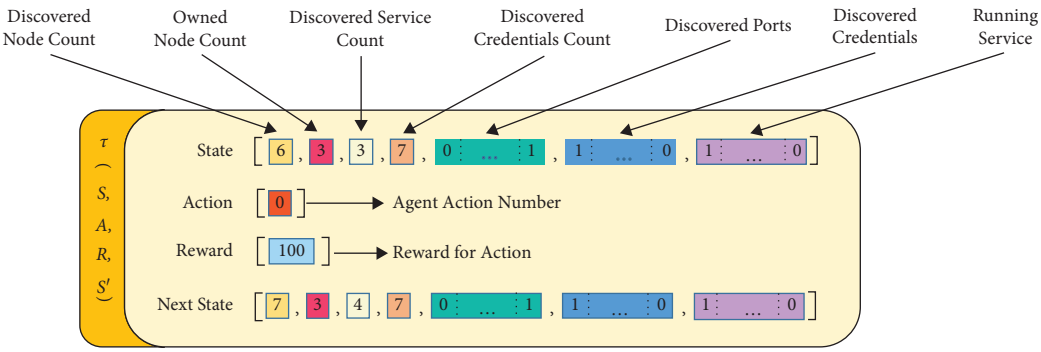

Figure 5: Structure of transition data.

Table 1: Action execution outcome and evaluation.

Outcomes	Evaluation
Leak new nodes	10
Leak credentials	5
Successful connection	20
Privilege promotion	10
Failed to exploit	−10
Repetitive action	−5

of the transformed state. The transfer function is unknown when formalising the PT process as an MDP.

5. Experiment

First, we build PT simulation network scenarios on the CBS platform developed by Microsoft. Second, we build an expert knowledge base containing transition data for multiple network scenarios by using the expert knowledge collection method introduced in Section 3. Finally, to validate the effectiveness of our proposed method, we use the DQfD algorithm and the DQN algorithm to perform PT path planning under scenarios containing elements of network defence deception (equipped with honeypots), respectively, and the performance of the algorithms is evaluated by specific metrics.

5.1. Main Experimental Procedures

5.1.1. Experimental Platform. Our experiments were conducted on the CyberBattleSim (CBS) platform developed by the Microsoft security team. CBS is an experimentation research platform to investigate the interaction of automated agents operating in a simulated abstract enterprise network environment. The simulation provides high-level abstraction

of computer networks and cyber security concepts. We construct a simulated network scene through CBS and encapsulate this network scene into a gym environment where we can interact with an agent and further combine it with RL-related algorithms for our experiments.

5.1.2. Network Scenario Building. We abstracted a real-world enterprise network scenario and built a simulated network scenario containing a honeypot based on the CBS platform. The topology of the network scenario is shown in Figure 6.

The simulated enterprise network scenario consists of a DMZ zone, Trust-1 zone, and Trust-2 zone. The Trust-1 and Trust-2 zones provide web services and database services in the enterprise network. We have deployed honeypots in two separate trust areas. Honeypots are replicas of sensitive servers and hosts and provide some common services, open more sensitive ports, and contain some invalid resources and information. Honeypots are set up to consume the attacker's resources and to mitigate the impact of the attacker's actions on the enterprise network. This is a high fidelity construction of a real-world network scenario. The firewall between each zone controls the access policy between zones. The host configuration information and firewall access policies and vulnerability information for the simulated enterprise network scenario are shown in Tables 2–4, respectively.

5.1.3. Penetration Testing Goals. In the simulated enterprise network scenario, the attacker has initially gained control of SpringBoot in the DMZ zone. The attacker uses this as a springboard machine for further lateral move. The goal of the PT process is to gain access to the control commands of the database server in the Trust-2 area in order to get further access to sensitive data and critical information. At the same time, the agent needs to avoid getting caught in the honeypot in the trust area. That is, the agent expects to obtain as many cumulative rewards as possible at the least cost of consumption.

5.1.4. Expert Knowledge Collection. For the construction of the expert knowledge base, we convert the artificial experience of successfully conducting PT into demonstration transition data that can be understood and learnt by the agent. In our experiments, we collected 1000 expert demonstration data from each of 10 different structured network scenarios and pushed them into the expert knowledge base. The scale of these network scenarios is all within a certain size range. We preplace these expert demonstration data in the demonstration data areas of the sum tree before pretraining.

5.1.5. Evaluation Metrics

(i) Average cumulative reward: In the process of applying the RL algorithm to train an agent for penetration testing, the cumulative reward earned in each round can directly indicate the training of the current round as the number of steps increases. Therefore, the average cumulative reward over multiple rounds can effectively show the average performance of the agent throughout the whole training process.

(ii) Probability of attacking honeypots: The honeypots in the simulated network scenario are hosts or servers with a cyber deception defence role. We calculate the number of times the honeypot is attacked in each round as a percentage of the number of actions performed by the agent. This metric assesses the effectiveness of expert demonstration data for policy training by the agent.

5.2. Experimental Results and Analysis. We trained the agent to perform PT using two algorithms, DQfD and DQN, respectively. The hyperparameter settings for the algorithms and the DQfD-specific parameter settings are shown in Tables 5 and 6, respectively.

The average of the cumulative rewards obtained by the agent over the 200 round episodes of training was counted to compare the performance of DQfD and DQN. Second, to verify the effectiveness of the incorporated expert knowledge, we counted the probability of attacking the honeypot for Honeypot-1 and Honeypot-2, respectively. To ensure the credibility of the experimental results, we conducted 10 experiments in the same network scenario. We plotted the average results of the 10 experiments as graphs as shown in Figures 7–9.

As can be seen from the experimental results in Figure 7, the DQfD algorithm incorporating expert knowledge is able to achieve the PT goal in fewer steps compared to the DQN algorithm (DQfD within 500 steps and DQN within 3000 steps). Redundant repetitions, exploitation failures, and actions that fall into the honeypot during the penetration test often result in penalties. Therefore, the larger the reward value accumulated in each round, the more it reflects the superiority of the agent's PT path and action selection strategy. The DQfD algorithm accomplishes the goal faster while earning more cumulative rewards in each round. The experimental results indicate that the DQfd algorithm improves the performance of penetration testing to a certain extent while demonstrating the superiority of fusing expert knowledge.

The results in Figures 8 and 9 show that intelligence trained using the DQfD algorithm has a significantly lower probability of attacking the honeypot hosts in the Trust-1 region and Trust-2 region in each episode. Compared to DQN, DQfD maintains a lower probability of attack throughout the training convergence, always below 0.1. The reason for fluctuations in DQfD in the early stages is due to the fact that in the early stages, the exploration rate ϵ is in the process of decaying. However, there is still a high probability of random exploration of actions. DQN has a high probability of attacking the honeypot in the early stages. As training progresses, trial-and-error experience is learned into the network model though. However, due to the lack of guidance from expert knowledge, its ability to avoid

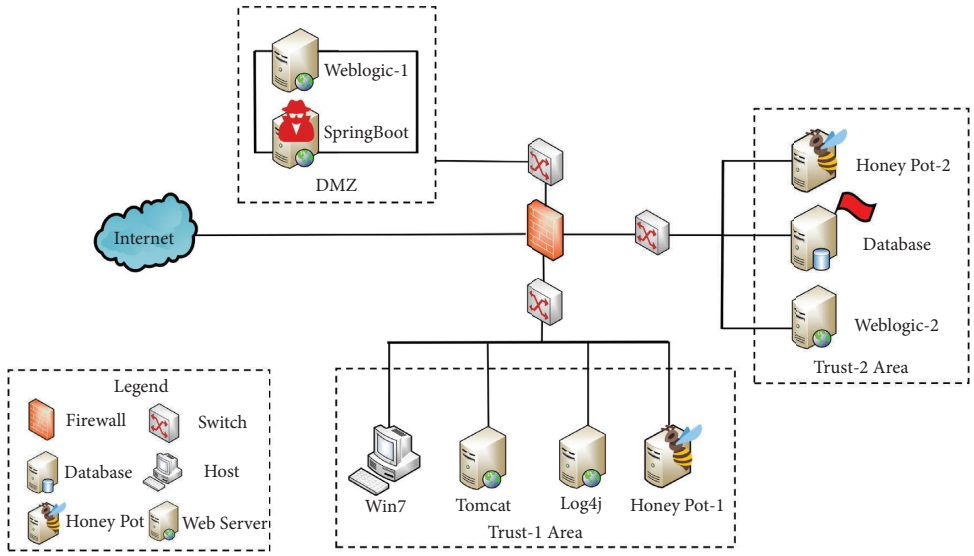

Figure 6: Enterprise network scenarios with honeypots.

Table 2: Host configuration information of the enterprise network.

Host ID	Vulnerability	Service	Open port	Value
Weblogic-1	CVE-2015-4852	HTTP	7001,5556	10
SpringBoot	Scancachehistory	HTTPS	8082	50
Windows7	CVE-2021-42287, scancachehistory	RDP, HTTPS, SMTP, FTP	8080, 3389, 25, 21	100
Tomcat	CVE-2017-12615	HTTP, SSH	8088, 22	100
Log4j	CVE-2021-44228, scancredhistory	HTTPS, RDP	3389, 8080	100
Honeypot-1	CVE-2016-10009	HTTPS, MySQL, FTP, SMTP, SSH	8080, 3306, 21, 25, 22, 2222	−200
Honeypot-2	CVE-2016-10009	HTTPS, MySQL, FTP, SMTP, SSH	8080, 3306, 21, 25, 22, 2222	−200
Database	CVE-2017-4971	HTTPS, MySQL, FTP	3306, 8080, 21	200
Weblogic-2	CVE-2015-4852	HTTP	7001, 5556	100

Table 3: Firewall policies for the enterprise network.

Sources	Destination	Service	Rule
SpringBoot	Windows7	HTTPS, RDP	Permit
SpringBoot	Tomcat	HTTP, SSH	Permit
SpringBoot	Log4j	HTTPS, RDP	Permit
SpringBoot	Honeypot-1		All permit
SpringBoot	Trust-2		All deny
Trust-1	Trust-2	HTTPS, MySQL, FTP, HTTP, SSH	Permit
Trust-1	DMZ		All permit
Trust-2	Windows7	HTTPS, RDP	Permit
Trust-2	Tomcat	HTTP, SSH	Permit
Trust-2	Log4j	HTTPS, RDP	Permit
Trust-2	Honeypot-1		All permit

deception defences was not significantly improved compared to DQfD.

The experimental results effectively reflect the role of incorporating expert knowledge in the identification and evasion of the deception defence components of the scenario during the PT performed by the agent. The less the intelligence interacts with the honeypot in each episode, the less the cost of completing PT will be consumed. This will greatly weaken the role of honeypot deployments from another perspective, where expert knowledge can guide and modify the agent's PT strategy.

Table 4: Vulnerability information and exploitation results.

Vulnerability	Type	Outcome	Cost
CVE-2015-4852	Remote exploit	Privilege promotion	10
CVE-2016-10009	Remote exploit	Privilege promotion	10
CVE-2017-12615	Remote exploit	Privilege promotion	10
CVE-2021-44228	Remote exploit	Privilege promotion	10
CVE-2021-42287	Remote exploit	Privilege promotion	10
Scancachehistory	Local exploit	Live nodes leaked	5
Scancredhistory	Local exploit	Credential leaked	5

Table 5: Hyperparameter setting of the algorithm.

Hyperparameter	DQfD	DQN
Batch size	512	512
Epsilon	0.9	0.9
Discount factor	0.015	0.015
Epsilon exponential decay	5000	5000
Epsilon minimum	0.1	0.1
Learning rate	0.01	0.01
Demonstration memory size	1000	*
Replay memory size	5000	5000
Target network update frequency	6	6
Max steps per episode	3000	3000
Training episode	200	200

Table 6: Special hyperparameter setting of the DQfD.

Hyperparameter	Value
Pretrain step	1000
N-step return weight λ_1	1.0
Supervised loss weight λ_2	1.0
L_2 regularisation weight λ_3	1.0
Expert margin $l(a_E, a)(a \neq a_E)$	0.8
N of N-step return	10
Prioritized replay exponent α	0.4
Prioritized replay constants ϵ_a	0.001
Prioritized replay constants ϵ_d	1.0
Prioritized replay importance sampling exponent β_0	0.6

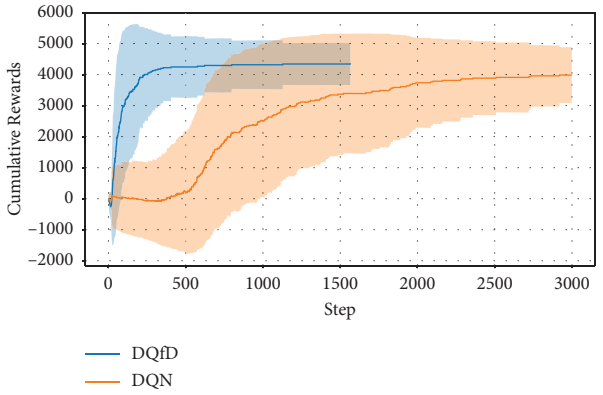

Figure 7: Changes in the average cumulative reward value.

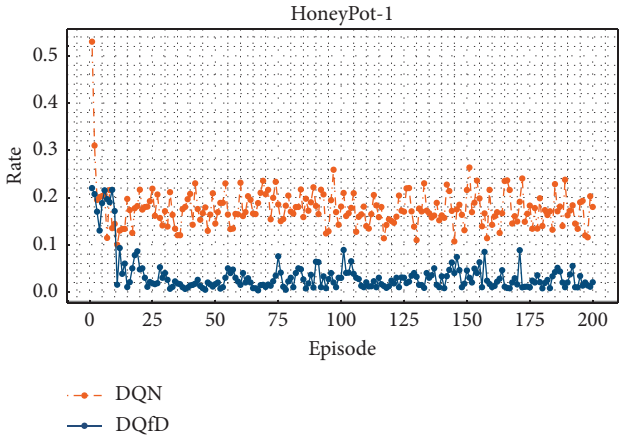

Figure 8: Probability comparison of attacking Honeypot-1.

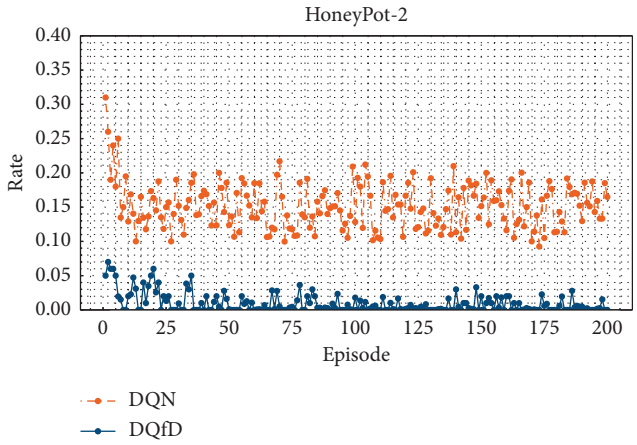

Figure 9: Probability comparison of attacking Honeypot-2.

6. Conclusion and Future Work

In this study, we focus on the effective use of expert knowledge in the process of intelligent PT. In order to address the poor interpretability of expert knowledge and the overfitting of algorithms to expert knowledge that exists in the current study, we propose an intelligent PT framework named DQfD-AIPT. DQfD-AIPT incorporates the collection of expert knowledge and guides the agent in the enhancement of PT strategies. The expert knowledge that we use is interpretable and generalisable for PT processes in network scenarios at a fixed scale. At the same time, we provide a detailed description of the process of transforming and collecting expert knowledge and define the structure of expert knowledge. Our proposed DqfD-AIP framework is generic and feasible. In terms of algorithms for combining expert knowledge, we use the DQfD method based on the RLED framework instead of traditional imitation learning.

The advantage of using the DQfD algorithm is that it combines the advantages of supervised and unsupervised learning. The DQfD algorithm makes reasonable use of the transition data from experience replay and avoids the phenomenon of overfitting of expert data. The results of experiments conducted in a network scenario with honeypots also indirectly indicate the validity of the expert demonstration data. Experimental results also show that the DQfD algorithm not only achieves higher cumulative reward values faster in network scenarios with honeypots but also makes better use of the expert demonstration data to avoid getting trapped in honeypots compared to DQN.

The efficiency and difficulty of PT depends on the complexity of the target network structure. Most of the expert knowledge that we currently collect is gathered through training in network scenarios of a specified size range. The coverage of expert knowledge is therefore relatively small and limited by the representational form of the transition data. In future research, we consider improving the interpretability of expert knowledge by better converting human PT experience into knowledge that can be accepted and learned by an agent. [15, 27, 28].

Data Availability

As a result, the data for this experiment were generated through training on the CBS platform, and no previous data were used. The transition data used in this article are generated through self-constructed scenarios.

Additional Points

Our experiment was carried out on the basis of Microsoft's open source CBS platform (CyberBattleSim). CBS highly abstracts the process of penetration testing from the network scene in the real world and builds simulated network scenarios. The agent algorithm interface is provided in CBS, and the agent uses the interaction with the environment and the reinforcement learning algorithm to obtain the penetration test strategy. The reinforcement learning method is an unsupervised machine learning method and difficult to repeat, and the data for simultaneous training are generated in the process of interaction between the agent and the concrete environment. The difficulty of penetration testing depends on the complexity of network scenarios. In reality, network scenarios are diverse, and the generation of transition data is also related to the structure of network scenarios.

Conflicts of Interest

There are no conflicts of interest regarding the publication of this paper.

References

[1] A. Chowdhary, D. Huang, J. S. Mahendran, D. Romo, Y. Deng, and A. Sabur, "Autonomous security analysis and penetration testing," in *Proceedings of the 2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, IEEE, Tokyo, Japan, December 2020.

[2] Y. Zhang, "Domain-independent intelligent planning technology and its application to automated penetration testing oriented attack path discovery," *Electron. Inf. Technol*, vol. 42, pp. 2095–2107, 2020.

[3] M. A. Abd-Elmagid, N. Pappas, H. S. Dhillon, and H. S. Dhillon, "On the role of age of information in the Internet of Things," *IEEE Communications Magazine*, vol. 57, pp. 72–77, 2019.

[4] Z. Fang, J. Wang, Y. Ren, Z. Han, H. V. Poor, and L. Hanzo, "Age of information in energy harvesting aided massive multiple access networks," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 5, pp. 1441–1456, 2022.

[5] F. Baiardi, "Avoiding the weaknesses of a penetration test," *Computer Fraud & Security*, vol. 2019, Article ID 10.1016/s1361-3723(19)30041-7, pp. 11–15, 2019.

[6] F. Holik, "Effective penetration testing with Metasploit framework and methodologies," in *Proceedings of the 2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI)*, IEEE, budapest, Hungary, November 2014.

[7] N. Polatidis, E. Pimenidis, M. Pavlidis, S. Papastergiou, and H. Mouratidis, "From product recommendation to cyber-attack prediction: generating attack graphs and predicting future attacks," *Evolving Systems*, vol. 11, pp. 479–490, 2020.

[8] M. Sultana, A. Taylor, and L. Li, "Autonomous network cyber offence strategy through deep reinforcement learning," *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, SPIE, vol. 11746, , 2021.

[9] J. Schwartz and K. Hanna, "Autonomous penetration testing using reinforcement learning," 2019, https://arxiv.org/abs/1905.05965.

[10] S. Zhou, J. Liu, D. Hou, X. Zhong, and Y. Zhang, "Autonomous penetration testing based on improved deep q-network," *Applied Sciences*, vol. 118823 pages, 2021.

[11] Zennaro, F. Massimo, and L. Erdodi, "Modeling penetration testing with reinforcement learning using capture-the-flag challenges: trade-offs between model-free learning and a priori knowledge," 2020, https://arxiv.org/abs/2005.14165, Article ID 12632.

[12] J. Chen, "GAIL-PT: a generic intelligent penetration testing framework with generative adversarial imitation learning," 2022, https://arxiv.org/.

[13] J. A. Bland, M. D. Petty, T. S. Whitaker, K. P. Maxwell, and W. A. Cantrell, "Machine learning cyberattack and defense strategies," *Computers & Security*, vol. 92, Article ID 101738, 2020.

[14] K. Qian, "Ontology and reinforcement learning based intelligent agent automatic penetration test," in *Proceedings of the IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, IEEE, Dalian, China, June 2021.

[15] D. Stiawan, "Cyber-attack penetration test and vulnerability analysis," *International Journal of Online and Biomedical Engineering*, vol. 13, 2017.

[16] H. M. Z. A. Shebli and B. D. Beheshti, "A study on penetration testing process and tools," in *Proceedings of the 2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pp. 1–7, May 2018.

[17] M. C. Ghanem and T. M. Chen, "Reinforcement learning for efficient network penetration testing," *Information*, vol. 11, no. 1, 6 pages, 2019.

[18] C. Sarraute, O. Buffet, and J. Hoffmann, "Penetration testing==POMDP solving?," 2013, https://arxiv.org/abs/1306.4714.

[19] J. Schwartz, H. Kurniawati, and E. El-Mahassni, "Pomdp+ information-decay: incorporating defender's behaviour in autonomous penetration testing," in *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 235–243, Singapore, June 2020.

[20] Z. Hu, R. Beuran, and Y. Tan, "Automated penetration testing using deep reinforcement learning," in *Proceedings of the 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, September 2020.

[21] K. Durkota and V. Lisý, *Computing Optimal Policies for Attack Graphs with Action Failures and Costs*, STAIRS, Geneva,Switzerland, 2014.

[22] J. Hoffmann, "Simulated penetration testing: from "dijkstra" to "turing test++," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 25, pp. 364–372, Jerusalem Israel, June 2015.

[23] A. Kanervisto, J. Pussinen, and V. Hautamäki, "Benchmarking end-to-end behavioural cloning on video games," in *Proceedings of the 2020 IEEE conference on games (CoG)*, IEEE, Osaka, Japan, August 2020.

[24] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: challenges, methods and progress," *Artificial Intelligence*, vol. 297, Article ID 103500, 2021.

[25] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: a survey of learning methods," *ACM Computing Surveys*, vol. 50, pp. 1–35, 2018.

[26] T. Hester, M. Vecerik, O. Pietquin et al., "Deep q-learning from demonstrations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, New Orleans LA USA, Febraury 2018.

[27] Y. Fenjiro and H. Benbrahim, "Deep reinforcement learning overview of the state of the art," *Journal of Automation, Mobile Robotics and Intelligent Systems*, vol. 12, no. 3, pp. 20–39, 2018.

[28] V. Mnih, *Playing Atari with Deep Reinforcement Learning*, https://arxiv.org/abs/1312.5602, 2013.