electronics

MDPI

Article

An Automated Penetration Testing Framework Based on Hierarchical Reinforcement Learning

Hongri Liu [1,2], Chuhan Liu [3,4], Xiansheng Wu [1], Yun Qu [5,*] and Hongmei Liu [6,*]

[1] School of Computer Science and Technology, Harbin Institute of Technology, Weihai 264209, China; liuhr@hit.edu.cn (H.L.); wuxiansheng@sf-express.com (X.W.)
[2] Weihai Cyberguard Technologies Co., Ltd., Weihai 264209, China
[3] Faculty of Computing, Harbin Institute of Technology, Harbin 150001, China; 23b903082@stu.hit.edu.cn
[4] Shandong Key Laboratory of Industrial Network Security, Harbin Institute of Technology, Weihai 264209, China
[5] Network Center, Harbin Institute of Technology, Weihai 264209, China
[6] China Mobile Group Shandong Co., Ltd., Jinan 250013, China
* Correspondence: quluan@hit.edu.cn (Y.Q.); liuhongmei@sd.chinamobile.com (H.L.); Tel.: +86-1506-314-2116 (Y.Q.); +86-1395-311-0072 (H.L.)

Abstract: Given the large action space and state space involved in penetration testing, reinforcement learning is widely applied to enhance testing efficiency. This paper proposes an automatic penetration testing scheme based on hierarchical reinforcement learning to reduce both action space and state space. The scheme includes a network intelligence responsible for specifying the penetration host and a host intelligence designated to perform penetration testing on the selected host. Specifically, within the network intelligence, an action-masking mechanism is adopted to shield unenabled actions, thereby reducing the explorable action space and improving the penetration testing efficiency. Additionally, the host intelligence employs an invalid discrimination mechanism, terminating testing after actions that do not alter system states, thereby preventing sudden increases in the number of neural network training steps for an action. An optimistic estimation mechanism is also introduced to select penetration strategies suited to various hosts, preventing training crashes due to value confusion between different hosts. Ablation experiments demonstrate that the host intelligence can learn different penetration strategies for varying penetration depths without significant fluctuations in training steps, and the network intelligence can coordinate with the host intelligence to perform network penetration steadily. This hierarchical reinforcement learning framework can detect network vulnerabilities more quickly and accurately, significantly reducing the cost of security policy updates.

Keywords: penetration testing; hierarchical reinforcement learning; action masking; invalid discrimination mechanism; optimistic value estimation; Markov decision process

Citation: Liu, H.; Liu, C.; Wu, X.; Qu, Y.; Liu, H. An Automated Penetration Testing Framework Based on Hierarchical Reinforcement Learning. *Electronics* **2024**, *13*, 4311. https://doi.org/10.3390/electronics13214311

Academic Editor: Chang Wook Ahn

Received: 13 August 2024
Revised: 24 October 2024
Accepted: 30 October 2024
Published: 2 November 2024

1. Introduction

Industrial control network environments, characterized by open system interconnection and high real-time requirements, face significant risks such as data leakage, ransomware, and zero-day attacks. These threats can result in severe data breaches and financial losses. Unlike traditional networks that primarily focus on identifying and blocking external attacks, industrial control networks employ penetration testing to proactively detect and repair vulnerabilities. Penetration testing gathers system information and exploits vulnerabilities from the perspective of attackers. It is particularly suitable for assessing the security of systems exposed to high-risk network environments where sustained attacks occur frequently [1]. Automated penetration testing (APT), using tools such as APT2 and AutoSploit, automatically cleans traces and records behaviors by maintaining a penetration load library and driving the intelligent penetration testing model [2]. This testing method offers the advantages of low costs, high speed, and ease of operation.

Reinforcement learning is the most promising technique used in penetration testing due to its ability to interact directly with the environment. APT based on reinforcement learning follows the process outlined below. Firstly, it understands the characteristics and vulnerability information of the target system and describes the penetration scenario using methods such as attack trees and attack graphs [3,4]. Secondly, it determines the optimal penetration paths and strategies. This involves using deterministic planning methods such as planning diagrams, or non-deterministic methods such as Markov models, to select the penetration path based on the real-time network state, intelligently selecting effective exploit loads and configure load parameters [5]. Next, a knowledge database is constructed to provide support for penetration technologies and tools, and an appropriate representation is chosen to characterize the vast amount of penetration information. Finally, APT makes penetration decisions and becomes involved in action planning [6]. During this process, agents take the next step by observing new states and rewards, constantly optimizing the strategy to adapt to different penetration test environments and ensuring that the system can automatically execute the most beneficial behaviors. The reinforcement learning enables adaptive and autonomous penetration testing, thereby improving the efficiency and accuracy of the tests.

However, as the network scale increases, APT based on reinforcement learning reveals several challenges. The state space is typically represented by vectors, with the dimension of the vector correlating to the number of hosts and their configuration information. The state space expands exponentially as the network scale grows, severely limiting the efficiency of reinforcement learning algorithms. Additionally, the intelligence cannot determine the type and number of vulnerabilities in each host, leading to its selection of all possible exploits. As the number of vulnerabilities increases, the action space of reinforcement learning also expands. This expansion increases the intelligence's optional space at each step so that the path searches become increasingly difficult. Furthermore, the intelligence can only obtain positive rewards when it successfully gains permissions from sensitive hosts during penetration testing, making the algorithm difficult to converge.

To address the dimension disaster in reinforcement learning for APT, hierarchical reinforcement learning decomposes a complex task into subtasks at different levels of abstraction. These subtasks reduce the action space, making the approach more suitable for the penetration testing of large-scale networks [7]. The hierarchical reinforcement learning process primarily employs state abstraction [8], temporal abstraction [9], and state space decomposition based on Markov decision processes [10]. It breaks down continuous state changes into intermittent state changes, effectively handling tasks that require multiple time steps to accurately reflect their action values.

The main contributions of this paper can be summarized as follows:

1. Penetration testing involves distinct tasks in the network and host layers. This paper presents a hierarchical dual-intelligence framework consisting of network intelligence and host intelligence. The network intelligence assigns penetration targets and subnet scanning operations to the host intelligence; the host intelligence executes penetration testing on multiple hosts. These two intelligence collaborate automatically and hierarchically, significantly reducing both the state space and the action space.
2. Network penetration testing is a continuous process of discovering new targets, and operations related to undiscovered targets should not be selected by the intelligence. We employ the action-masking mechanism to exclude unenabled actions, thereby narrowing the actual action space and reducing the exploration burden on intelligence.
3. Different hosts have varying levels of vulnerability. Previous studies often assumed that the target host could be either vulnerable or fully accessible, and the penetration testing is terminated upon reaching a fixed target or timeout. This paper introduces the active return actions and the optimistic estimation mechanism, allowing the intelligence to conclude the test promptly after achieving the maximum depth of penetration. These strategies are particularly suited for targets where the depth of penetration cannot be predetermined.

4. Only a small number of the identified host actions are valid. To stabilize the training of the intelligence, we assess the validity of actions and modify the state transition graph to prevent invalid actions from creating self-loops.

The remainder of this paper is organized as follows: Section 2 discusses related works on penetration testing utilizing both general and hierarchical reinforcement learning. Section 3 introduces the general penetration testing model. In Section 4, we design an innovative hierarchical reinforcement learning framework. Then, Section 5 presents the penetration testing results along with relevant analysis. Finally, some brief conclusions and notes on future work are given in Section 6.

2. Related Work

There are two main strategies for automatic penetration testing based on reinforcement learning. The first strategy is attack graphs, a white-box method that eliminates irrelevant actions to model the testing network [11–14]. In this approach, the attack graph rules need to be designed by collecting global knowledge before training the intelligence. The second strategy involves real network docking or simulation, which fully leverages reinforcement learning and environmental interactions. This method does not need global knowledge and only requires the intelligence to gather information.

Many scholars have achieved single-step attack learning on hosts [15–19], while NIG-AP technology uses information gain as a reward signal to facilitate multi-step attack learning [20]. In current studies on network-oriented penetration testing that employ various attack methods, the DQN (deep Q network) algorithm, which integrates multiple reinforcement learning techniques [21,22], is frequently used. Additionally, CRLA utilizes multiple intelligence to decompose actions, effectively managing the extensive action space [23]. Recently, Nguyen et al. embedded actions into a low-dimensional vector space and introduced the Wolpertinger architecture [24,25], which selects the most appropriate action by clustering similar actions. These established strategies assume that the system has a fixed penetration target and a clear path to that target, which is not always realistic and results in a vast state space.

In hierarchical reinforcement learning, there are two prevalent methods for managing long sequential decisions [26]. The first is option-based intelligence [27], which involves higher-level intelligence selecting different lower-level intelligence modules. For instance, the IAPTF framework divides a large network into smaller clusters and schedules options by using a planner algorithm [28]. The second method is the "goal-return" framework, which sets different goals at the higher level and achieves these goals at the lower level. Examples of this framework include H-DQN [29] and Feudal Networks [30].

Based on the methods mentioned above, the proposed algorithm DAA employs two-layer intelligence [31], consisting of a top layer and bottom layer. The top layer understands the network structure, while the bottom layer selects actions to exploit single-step vulnerabilities. Similarly, the iPathD algorithm [32], with dual-intelligence uses the top layer to select penetration targets and the bottom layer to conduct multi-step attacks. The advantage of these hierarchical reinforcement learning algorithms is their ability to effectively reduce the state and action space. However, like general reinforcement learning, they often assume a fixed target for penetration or a set duration. Such assumptions lead to inefficiencies and limit adaptability to varying penetration depths.

In summary, penetration testing involves large action spaces and long decision sequences. Existing studies on hierarchical penetration testing are still in the initial stages, often assuming scenarios with feasible paths or the complete action space of Gray-Box Testing. Most related research directly incorporates the latest advancements without thoroughly examining the unique characteristics of penetration testing. To address these gaps, this paper investigates the process of new host discovery to differentiate between enabled and unenabled actions, helping to develop training methods that are adaptable to different network sizes. Specifically, we establish the invalid discrimination mechanism to stabilize training. Additionally, we explore multi-host scenarios with varying degrees of penetra-

tion, implementing return actions and the optimistic estimation mechanism for multi-host policy learning.

3. Penetration Testing Model

The system performing penetration testing generally assumes that the attacker can directly connect to the OT network by bypassing the firewall through switches, reaching target operating machines, office machines, and file servers. Most penetration testing models constructed for this ICS environment utilize the Markov decision process to predict the transformation of the system state following the executed action. During these penetration testing processes, Deep Q Networks and TD error updates are employed to minimize training errors.

3.1. Markov Decision Process

The Markov decision process (MDP) serves as a theoretical foundation for reinforcement learning, providing a mathematical framework for sequential decision-making in fully observable environments [33]. The MDP is defined by five components (S, A, R, T, γ):

S represents the state space, encompassing all possible states of the system.

A represents the action space, containing all possible actions that transition the system to the next state.

R is the reward function, which assigns a reward r to the intelligence when the system transitions to the desired state.

T describes the transition probability of moving to the next state s' after taking action a in current state s.

γ is the discount factor used to calculate the cumulative discounted reward. It is calculated as follows:

$$r = R\left(s, s', a\right), \tag{1}$$

A cumulative discounted reward is the sum of discounted action rewards from the current state to the end state. It is calculated as the following equation:

$$return = \sum_k \gamma^k r_k, \tag{2}$$

The MDP model's state transitions under the action control, with the cumulative discounted reward reflecting the long-term effects of action decisions.

Figure 1 illustrates a network penetration testing scenario where the attacker located in host user 1 traverses the IT network to reach the OT network and subsequently launches attacks on the OT server. Initially, the attacker conducts active or passive investigation on the target network and analyzes the penetration testing framework. Based on the identified vulnerabilities, the attacker designs a strategy to bypass detection using techniques such as obfuscation or covert channels. After gaining initial access to the target system, the attacker lifts power and maintains long-term control over system resources while ensuring stealth. The attacker then proceeds with lateral movement across the network, collecting and exfiltrating sensitive data. Finally, upon completing the attack, any operational traces are cleared and the impact is evaluated, thereby avoiding detection by future penetration testing frameworks.

The state space for the penetration intelligence applicable to this specific network comprises states with reserved hosts. Each host's state information includes details such as open ports, service categories, operating systems, acquired user rights, system rights, and the current running status of the OT service. When information is unknown, its flag will be marked as -1.

The action space consists of all possible actions of hosts at the network level. It includes subnet scanning for discovering active hosts, port scanning, vulnerability exploitation, local privilege escalation, and OT services being affected. Depending on the specific services and operating systems, multiple options are available for vulnerability exploitation and local privilege escalation.

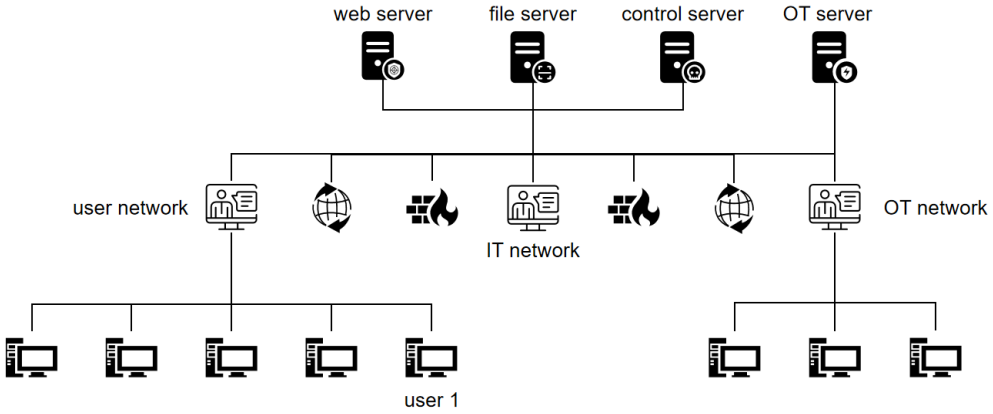


Figure 1. Network penetration testing scenario.

The action intended to be performed when a system transits from the current state to the next is termed the target action. Positive rewards are assigned when the target action is successfully executed, while negative rewards, serving as penalties, are given for other invalid actions.

Penetration testing consists of two key phases: the initial exploration and tentative execution on the target network, followed by the summarization and demonstration of the feasible path. Comprehensive penetration testing involves not only showcasing how information is gathered but also detailing the process of network penetration. The optimized penetration path is crucial as it directly reflects the outcomes of the testing, providing a clear representation of system risks and effectively capturing the client's attention. To enhance the application of penetration testing, a state integration interface has been designed to facilitate the collection of host information by the intelligence.

3.2. DQN Algorithm

The Deep Q-Network (DQN), a value-based deep reinforcement learning algorithm, combines Q-learning with deep neural networks [34]. A dual DQN (DDQN) incorporates a Q-estimation network and a Q-target network. Its dual Q-value network structure enhances the stability of the Q-value estimation [35]. Search algorithm models, including heuristic search methods, often fall into local optima in complex network environments, resulting in low search efficiency. To address this, the DDQN algorithm is employed as the penetration testing model in this paper, as it mitigates the value confusion problem that arises during training and accelerates the process of discovering the optimal strategy compared to general reinforcement learning models.

In the DDQN structure, the intelligence stores the experiences of interactions with the environment in an experience replay buffer, from which a portion of experience is randomly sampled during network updates. The Q-estimation network evaluates the Q-value for each action a_i, while the Q-target network computes the Q-value for the next state s_{i+1}. TD error is used to minimize the loss in the Q-estimation network, as described by the following equation:

$$loss = r + (1 - x)\gamma max_{a'} Q'(s_{i+1}, a') - Q(s_i, a\ ), \tag{3}$$

After iterative updates to the Q-estimation network, its parameters are copied to the Q-target network. Figure 2 illustrates the architecture of the DDQN.

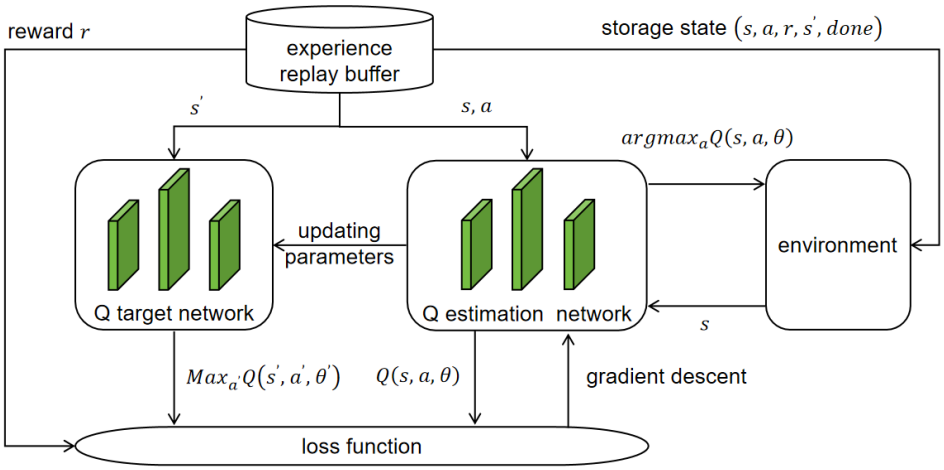


Figure 2. DDQN architecture.

4. Hierarchical Penetration Testing

In this section, we propose a hierarchical penetration testing framework with a dual-intelligence core to enhance the effectiveness of penetration testing. This framework consists of the network intelligence and the host intelligence. The network intelligence autonomously scans target systems for potential vulnerabilities and swiftly responds to emerging threats. It can adapt to the evolving security challenges in time, ensuring the spontaneity and immediacy of penetration testing. The observation space of the network intelligence comprises all the information it gathers from the environment, enabling the identification of weak subnets and nodes that may serve as potential attack targets. The action space of the network intelligence defines the operations it can perform at the network level, facilitating network exploration and the discovery of potential vulnerabilities. The host intelligence conducts penetration testing for various types of systems and continuously monitors the security status of the host systems. The observation space of the host intelligence focuses on the status information of the devices it controls, aiding in the assessment of the host's security and determining which penetration actions can be executed. The action space encompasses all actions that can be performed at the host level, targeting the system and application layers to achieve persistent control. This timely detection of new vulnerabilities ensures the diversity and sustainability of penetration testing.

The network intelligence employs the action masking mechanism to interact with the experimental environment, while the host intelligence utilizes the optimistic estimation mechanism to support the network intelligence. The invalid action discrimination mechanism leverages a loss function to differentiate between actions that leave system in original state after execution and normal actions. This mechanism effectively reduces the number of invalid actions performed by the host intelligence, significantly narrowing the state and action space. The optimistic estimation mechanism allows for the penetration testing without the need to assume a fixed penetration target or a duration. It maximizes the system's value by working in tandem with the active return action, enabling adaptability to varying penetration depths.

4.1. Hierarchical Penetration Testing Framework

To accommodate the characteristics of hierarchical reinforcement learning, the intelligence linkage scheme is designed with five layers. The overall hierarchical intelligence framework is illustrated in Figure 3. The first layer is the collection layer; it features an observation collection interface responsible for receiving and processing environmental information. The second layer, the integration layer, consists of a reward–judgment interface that relays reward information to the network intelligence and a state integration interface that passes integrated information to both intelligences. The third layer, known as the intelligence layer, encompasses the network intelligence and the host intelligence, forming

the core of the framework. The fourth layer is the transport layer; it organizes the output generated by the intelligence layer. Finally, the fifth layer, the execution layer, comprises an action execution interface that performs the executable actions and then outputs the actions to the environment. This dual-intelligence framework enables efficient environmental interaction and action execution by leveraging different intelligence components, thereby enhancing the framework's adaptability and responsiveness to potential threats.

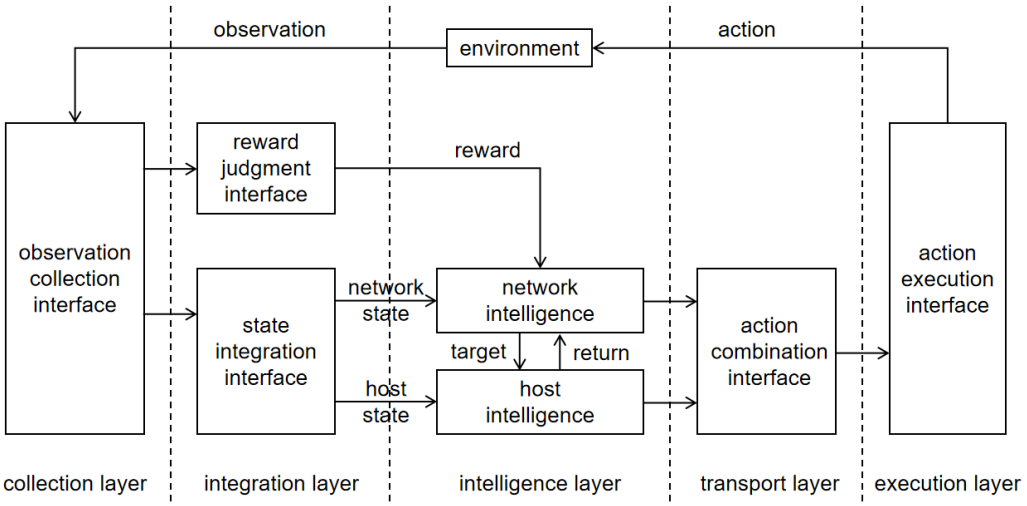


Figure 3. Hierarchical intelligence framework.

During the testing process, actions that cause state changes can be observed in the collection layer. These observations are processed and analyzed by the intelligence layer to determine the next action to be executed. The flow of information is detailed below.

The observation collection interface receives the results returned by the penetration tool after actions are executed. It subsequently extracts valuable information from fragments of the system's state, such as logs indicating the acquisition of user privileges.

The state integration interface consolidates historically recorded information fragments to provide a comprehensive representation of the testing network. This information includes details such as host identities, open ports, and other relevant data. The interface maintains a mapping table and assigns unique identifiers to new subnets and host IPs during the training. These identifiers remain consistent throughout the execution, preventing discrepancies that might arise from the varying discovery order of subnets and hosts. Each identifier is associated with a discovery flag, which indicates whether the object has been identified in the current action. At the beginning of each action, the flag is initialized to 0 and transmits to 1 when the object is discovered. The network state is determined by aggregating the known states of each numbered host, with a value of -1 assigned when the flag is not set. Fixed states represent constant hosts, aiding the intelligence in learning correlations between host information.

The state integration interface delivers integrated states to the network intelligence, which outputs actions based on the network state. When the network intelligence identifies a specific host target, the interface transmits the corresponding host states to the host intelligence. The host intelligence then selects an appropriate action based on the received host state.

The reward–judgment interface processes observations and generates reward signals based on predefined goals. A large positive reward is granted when actions successfully transition states to the expected are executed, indicating a successful impact on the OT service. Conversely, a small negative reward is granted when other invalid actions are executed.

Both intelligences utilize the DDQN algorithm. The action space for the network intelligence includes subnet scanning and host target determination. The number of scanning actions corresponds to the reserved subnets, while the number of target determination actions matches the reserved hosts. The network intelligence activates the host intelligence when the target is output. The design of the network intelligence will be detailed in Section 4.2.

The host intelligence receives host states from the state integration interface based on the outputs provided by the network intelligence. Its action space includes host penetration actions and active return actions. When the return action is selected, decision-making authority is handed back to the network intelligence. The design of the host intelligence will be detailed in Section 4.3.

The action–combination interface integrates the actions determined by both the network and host intelligence. If the output is related to a host, the action parameter will be the host IP from the host mapping table. Conversely, if the output is an action for subnet scanning, the subnet parameter will be determined based on the subnet mapping table.

The action–execution interface receives various types of parameters. It either invokes penetration tools to perform network penetration or operates the network simulator to execute actions.

4.2. The Network Intelligence

In hierarchical penetration testing experiments, the network intelligence interacts with the environment to assist the host intelligence in advancing the system to the next state. The network intelligence employs the action-masking mechanism during action execution to reduce the action space, significantly enhancing the efficiency of the experiments.

4.2.1. Environmental Interaction Method

This section introduces the selection of intelligence, the transmission of actions, and how the actions interact with the environment. Figure 4 illustrates the flowchart for obtaining and outputting actions. In this process, the flag indicates which intelligence is responsible for deciding the current action. It is important to note that the target action of the network intelligence and the return action of the host intelligence are virtual actions that do not interact with the environment. When the flag is set to 1, the action space comprises network-level actions that can be executed by the network intelligence, including subnet scanning, host targeting, and so on. Conversely, when the flag is set to 0, the action space consists of penetration actions, such as power lifting and vulnerability exploitation, as well as the active return actions performed by the host intelligence. The flag transitions from 1 to 0 after the network intelligence executes actions, enabling the host intelligence to make the next decision. If the host intelligence selects an actual penetration action, the output is sent to the environment. However, if the host intelligence opts for the active return action, decision-making power reverts to the network intelligence. To avoid deadlock caused by repeatedly selecting the network intelligence, the network intelligence employs the epsilon-greedy mechanism, which maintains a degree of randomness in the selection process. Additionally, a timeout mechanism is in place to trigger random actions if valid actions are not produced during decision switching.

The network intelligence interacts directly with the environment through subnet scanning, enabling immediate feedback and experience storage. The current state of the system is treated as the state of the network intelligence, which interacts with the environment through the host intelligence when targets are output. The actions of the network intelligence are not complete until the host intelligence executes an active return. In this scenario, the next state of the network intelligence is represented by the system state. The reward for the network intelligence is the cumulative sum of rewards earned during the interaction.

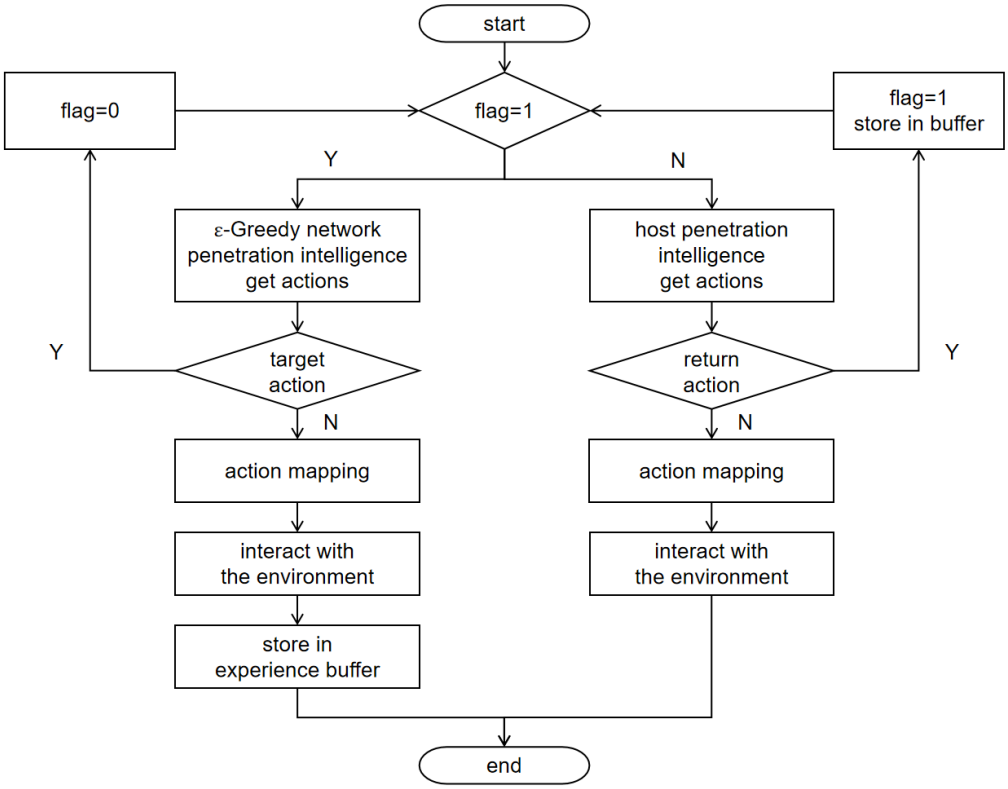


Figure 4. Action transmission flowchart.

4.2.2. Invalid Action Determination

In penetration testing, invalid actions occupy the majority of the action space. An invalid action is represented as a self-loop in the state transition diagram, as it does not result in a state change. The Q-values of invalid and valid actions are often similar, leading the neural network to make erroneous selections during training. Invalid actions are among the most common errors in the training of DQNs, potentially causing the intelligence to become trapped in an ineffective loop.

When an invalid action that leaves the state unchanged or returns a failure message occurs, the intelligence training process is terminated. In the DQN algorithm, the flag x is set to 1 in the experience and stored in the experience buffer. The Q-value network is then updated using the actual loss function, which is described by Equation (4).

$$loss = r - Q(s_j, a), \tag{4}$$

This mechanism adjusts the Q-value of the invalid action to be close to r, which is typically negative or zero, thereby clearly distinguishing invalid actions from valid ones. The determination process continues until a timeout occurs or the intelligence actively chooses to return.

The network intelligence training algorithm using invalid action determination is presented in Algorithm 1.

Algorithm 1 the network intelligence training

1: load the host penetration intelligence
2: initialize the experience repaly buffer D
3: initialize the Q-estimation neural network parameters θ_1 randomly, copy θ_1 to the Q-target neural network parameters θ_2
4: **for** each loop **do**
5: initialize the testing environment, obtain initial state s_t, initialize control to the network intelligence
6: **for** each step t **do**
7: obtain the action a_t with probability ε, obtain the action $a_t = argmax_a Q(s_t, a)$ with probability $1 - \varepsilon$
8: **if** a_t is not the return action **then**
9: execute the action, obtain the next status s_{t+1} and the reward r
10: initialize $x = 0$
11: **else**
12: initialize $x = 1$
13: **end if**
14: store experience $(s_t, a_t, r_t, s_{t+1}, x)$ in D
15: sample batch experience $(s_j, a_j, r_j, s_{j+1}, x)$ from D, calculate the error between $Q(s_j, a_j)$ and $r_j + (1 - x)\gamma max_{a'} Q'(s_{j+1}, a')$, backward propagate and update the parameters
16: **if** timeout or $x = 1$ **then**
17: jump out of loop
18: **end if**
19: **end for**
20: **end for**
21: decrease ε, copy θ_1 to θ_2 after a certain number of turns
22: **return**

Steps 8 to 13 incorporate the invalid action discrimination mechanism. Invalid actions are identified based on observation states that remain unchanged, meaning that, when the subsequent DQN training extracts experiences, the actual loss function is adjusted, leading to changes in the neural network's training parameters. The invalid action discrimination mechanism assigns a value of 0 to the execution of invalid actions.

As the scale of the tested network structure expands, the network intelligence adjusts the neural network parameters and retrains the model to ensure that the output layer aligns with the number of devices in the network.

The time complexity of the algorithm depends mainly on the number of episodes, so reducing the number of episodes required for convergence can enhance the efficiency of the training process. The optimized framework, incorporating both invalid action determination and the optimistic estimation mechanism, ensures enhanced training stability and faster convergence, making it highly suitable for deployment in large-scale penetration testing networks.

4.2.3. Action Masking Mechanism

This section introduces the action-masking mechanism, which is employed to eliminate identified invalid actions and reduce the exploration space.

The network intelligence provides a precise number of reserved actions, including fixed subnet scanning and targeted host actions, for the network output layer. When reserved actions in the action space are normally executed, the action–execution interface retrieves the IP of the target host and subnet either from a predefined target list or strategically based on the most recent penetration results of the host intelligence. The action-masking mechanism disables invalid actions in the action space and sets them as unavailable by omitting the flag for hosts or subnets in the state integration interface, thus reducing the action space. To accurately replicate the information gathering process and represent access relationships, the action–execution interface refrains from retrieving IP parameters and

executing actions against the target when masked actions are present. In such cases, failure information is returned directly through the action–execution interface, as the outcome is predictable and the action is rendered meaningless before selection.

Figure 5 illustrates the principle of the action-masking mechanism, which prevents the intelligence from selecting unenabled actions [36]. Specifically, the identity numbers and IP addresses of hosts are stored in a predefined mapping list that categorizes the information of the target subnet and host based on operating system types and open ports. The action mask is represented as a vector corresponding to specific actions within the action space of the network intelligence. When a host is bound to a specific IP in the mapping list, its flag is set to 1. The mask for the action targeting this host is set to 0, indicating the action is selectable, while the masks for other actions are set to 1, indicating those actions are blocked.

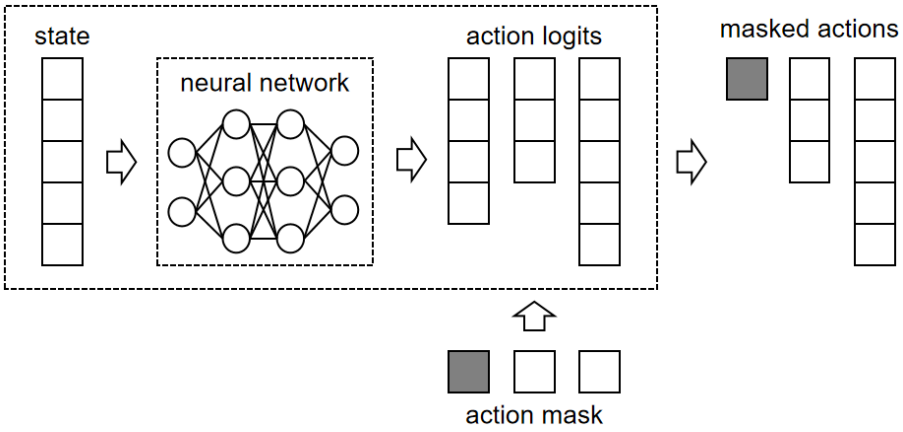


Figure 5. Action masking mechanism.

The DQN algorithm with the epsilon-greedy mechanism selects the action with the highest Q-value among the unblocked actions with the probability of 1 and randomly selects actions from the unblocked set with the probability of epsilon. The mask for the next state is stored in the experience replay buffer after the action is executed. When the network is updated, the maximum Q-value of the target network is chosen from the Q-values of the unmasked actions.

4.3. The Host Intelligence

The host intelligence must be capable of handling multiple penetration strategies and returns decision-making power to the network intelligence after completing sufficient penetration. The representative state transition diagram for host penetration testing is shown in Figure 6, where A1, A2, A3, A4, A5, and A6 represent actions. Only specific actions can advance penetration to particular host states. For instance, a system in the discovered state can only progress to the scanned state through action A1, as the actions A2, A3, A4, A5, and A6 are invalid for the system at that time. These invalid actions will leave the system in the discovered state.

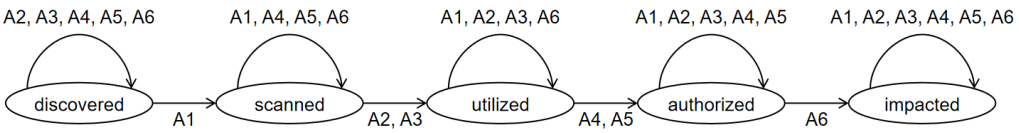


Figure 6. Example for penetration testing state transition diagram.

In the discovered state, the penetration testing collects information about script types, server types, and other relevant details to gain sufficient knowledge about the target program. In the scanned state, appropriate vulnerabilities are identified and exploited

based on the system type. In the utilized state, the testing process obtains intercept data transmitted by the target program and gains control over the target host. In the authorized state, the testing analyzes existing vulnerabilities within the program and provides effective solutions. Finally, in the impacted state, the penetration testing involves the deletion of system and program logs.

4.3.1. Active Return Action

Only a small number of hosts in the network have service vulnerabilities, and even fewer can be accessed due to the system privilege. Since hosts have varying depths of penetration, it is impractical to set a fixed penetration target. The active return action is introduced into the host action space to enable the host intelligence to explore hosts as thoroughly as possible and to terminate the process when further penetration is not feasible. This action allows the system to actively end exploration and return decision-making power to the network intelligence, which invokes actions at the appropriate time.

To guide the intelligence in determining when to use the active return action, positive rewards which incentivize the exploration are given for the first successful scanning, vulnerability exploitation, privilege escalation, and OT service impact. The host intelligence uses the active return action to terminate exploration when no additional rewards can be gained. The state transition diagram for host penetration with the active return action is shown in Figure 7. For example, a system in the discovered state progresses to the scanned state through action A1, while invalid actions A2, A3, A4, A5, and A6 will not leave the system in the original state. The active return action is actively employed when the host intelligence has not invoked any predefined actions and has thoroughly explored the hosts, ensuring that the test is terminated in a timely manner to prevent system crashes.

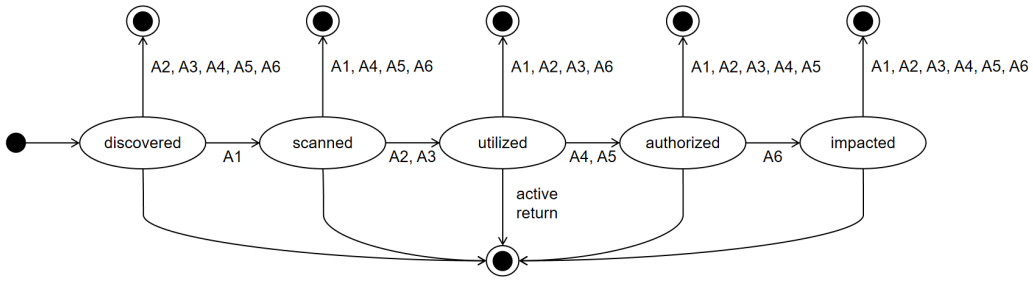


Figure 7. Penetration testing state transition diagram with the active return action.

4.3.2. Optimistic Estimation Mechanism

Different hosts have varying levels of penetration depths, yet they may share the same state representation. Hosts with identical open services can be exploited to gain user privileges. The ability to escalate from user to system privileges depends on the operating system version within the system information. As a result, the value-based reinforcement learning algorithm may assign different values to the same state when trained on different hosts. When these hosts are trained together, the value obfuscation leads to training failure.

To enable the intelligence to learn strategies compatible with multiple penetration testing scenarios, the optimistic estimation mechanism aligns the values of the same state across different hosts. The optimistically estimated value of a state represents the maximum potential value that can be achieved from the state. The system assigns a fixed estimation value based on the sequence of states: scanned, utilized, authorized, and impacted. This value indicates the potential of a system for successful privilege escalation and the ability to affect OT services, irrespective of the outcome. The actual state information is confirmed after the successful execution of authorization actions.

The host intelligence training algorithm using the optimistic estimation mechanism is presented in Algorithm 2.

Algorithm 2 the host intelligence training

1: load the host penetration intelligence
2: initialize the experience replay buffer D
3: initialize the Q-estimation neural network parameters θ_1 randomly, copy θ_1 to the Q-target neural network parameters θ_2
4: **for** each loop **do**
5: initialize the testing environment, obtain initial state s_t, initialize control to the network intelligence
6: **for** each step t **do**
7: obtain the action a_t with probability ε, obtain the action $a_t = argmax_a Q(s_t, a)$ with probability $1 - \varepsilon$
8: **if** a_t is not the return action **then**
9: execute the action, obtain the next status s_{t+1} and the reward r
10: initialize $x = 0$
11: **else**
12: initialize $x = 1$
13: **end if**
14: determine the value $V(s_{t+1})$ according to the next state s_{t+1}
15: store experience $(s_t, a_t, r_t, s_{t+1}, x, V(s_{t+1}))$ in D
16: sample batch experience $(s_j, a_j, r_j, s_{j+1}, x, V(s_{j+1}))$ from D, calculate the error between $Q(s_j, a_j)$ and $r_j + (1 - x)V(s_{j+1})$, backward propagate and update the parameters
17: **if** timeout or $x = 1$ **then**
18: jump out of loop
19: **end if**
20: **end for**
21: **end for**
22: **return**

Step 14 to 16 incorporate the optimistic estimation mechanism. Step 15 influences the recording of the experience buffer, while step 16 modifies the loss function during experience replay. The optimistic estimation mechanism serves as an estimate of the penetration state's value. In the five-step penetration model, it assumes that all machines will eventually be impacted. This value represents the expected future rewards. For any device, a specific state can be assigned an optimistically estimated value, which is directly linked to the penetration state within the state representation.

When the network scale expands, the underlying host intelligence remains unchanged and does not require retraining. Additionally, the time complexity of the algorithm primarily depends on the number of episodes, too.

5. Experimental Results and Discussion

The experiments were conducted using various network scenarios created by the CybORG network simulator [37]. In these experiments, we evaluated whether the improved training algorithm would crash and compared the jitter rate of our algorithm against the algorithm without the optimistic estimation mechanism, as well as the algorithm lacking the invalid action discrimination mechanism. The crash of the experiment indicates that the intelligence fails to identify the optimal penetration strategy, resulting in a lack of convergence during training. The jitter rate reflects the extent of score fluctuations after the training has converged. The results demonstrate that the network and host intelligence can effectively collaborate and that the optimized algorithm offers a superior training strategy.

5.1. Experimental Validation of the Host Intelligence

The host intelligence randomly encounters various hosts with different penetration depths during the training. The neural network parameters and training parameters are shown in Table 1.

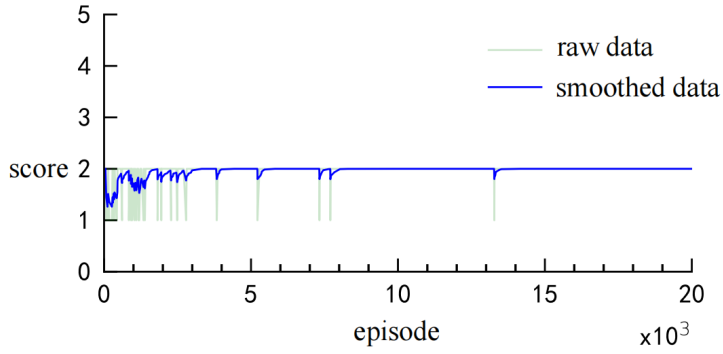
Table 1. The parameters of the host intelligence.

Parameter	Set Value
Q-target network and Q-estimation network	input layer: obs_dim interlayer: 64 output layer: actions_dim
gamma	0.9
batchsize	32
epsilon	1
end epsilon	0.05
Q-target network update interval	200
learning rate	0.01

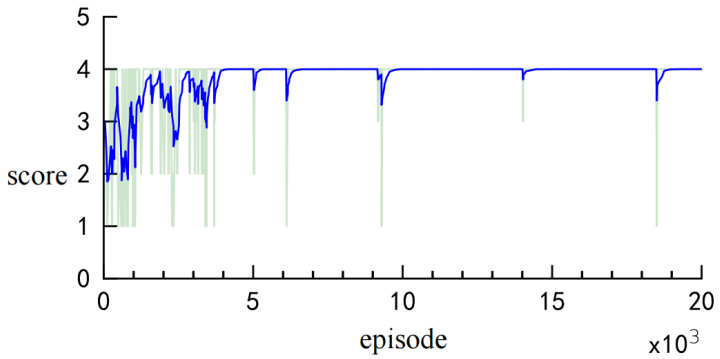
Figure 8 illustrates the training curves for three hosts. The horizontal axis represents the number of episodes and the vertical axis displays the scores. The green line represents the result of the raw data and the blue line corresponds to the smoothed data. The smoothed data refers to data that mitigate the jitter rate compared to the raw data, thereby facilitating a clearer observation of training trends. It is calculated using the smoothing algorithm integrated within the TensorBoard:

$$smoothed[i] = \frac{raw[i-1] \times weight + raw[i]}{weight + 1}, \tag{5}$$

where the weight is 0.99. Epsilon is maintained at 5% throughout all stages. Experimental curves demonstrate that the intelligence effectively learns strategies for hosts with different penetration depths by employing the active return action and the optimistic estimation mechanism. The 5% random exploration introduces minor jitters after the training reaches convergence. The invalid action discrimination mechanism ensures that the output of the neural network remains stable, facilitating the learning of the optimal strategy.



(**a**)



(**b**)

Figure 8. *Cont.*

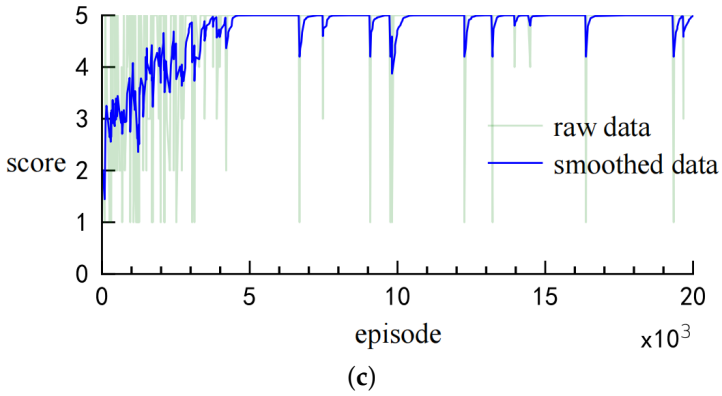


Figure 8. Training curves of the host intelligence in a one-host scenario. (**a**) Score of host0; (**b**) score of host1; (**c**) score of host5.

Figure 9 presents the training curves for host1 without the optimistic estimation mechanism. It demonstrates that the testing process fails in the later stages. This failure occurs because the training value is mixed up with values from other hosts, preventing the intelligence from accurately estimating action rewards. As a result, the score decreases and the training fails to converge, leading to the slow crash of the experiment.

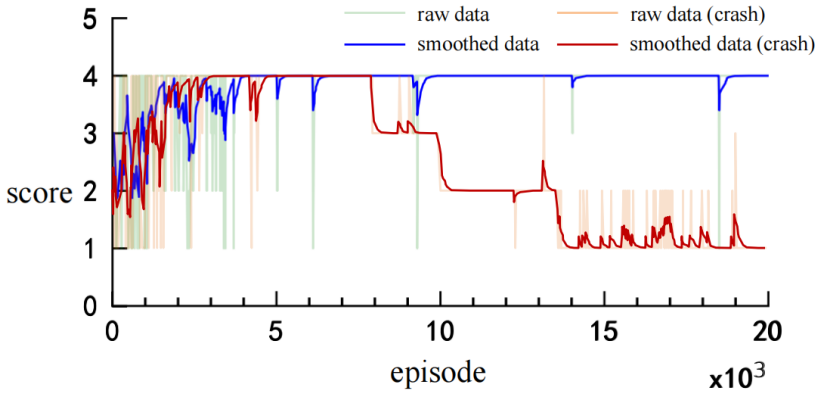


Figure 9. Crash curves of the host intelligence without the optimistic estimation mechanism.

The intelligence may experience fluctuations without the invalid action discrimination mechanism. Figure 10 presents the experimental results, where Figure 10a shows that curves experience significant fluctuations in most cases, and Figure 10b indicates a sharp increase in the number of penetration steps around the 1700th act. These results suggest that the presence of invalid actions causes instability during the training.

The experimental results demonstrate that the hierarchical penetration testing framework proposed in this paper significantly reduces costs and improves efficiency. Regardless of changes in real-world network architecture, the consumption of network resources remains consistent when utilizing the same neural network design. The time consumption in reinforcement learning is primarily influenced by the simulator's runtime. With the same number of training rounds, the use of the active return action enables the intelligence to actively terminate a training round, resulting in fewer steps being required for convergence. As shown in Figure 10, host3 requires approximately six steps to reach the training convergence, while it takes dozens of steps when the training fails to converge. The algorithm presented in this paper reduces the number of steps per round, ensuring a faster convergence and saving time. Additionally, it minimizes invalid actions, preventing the negative impact of excessive invalid attack traffic on the network environment.

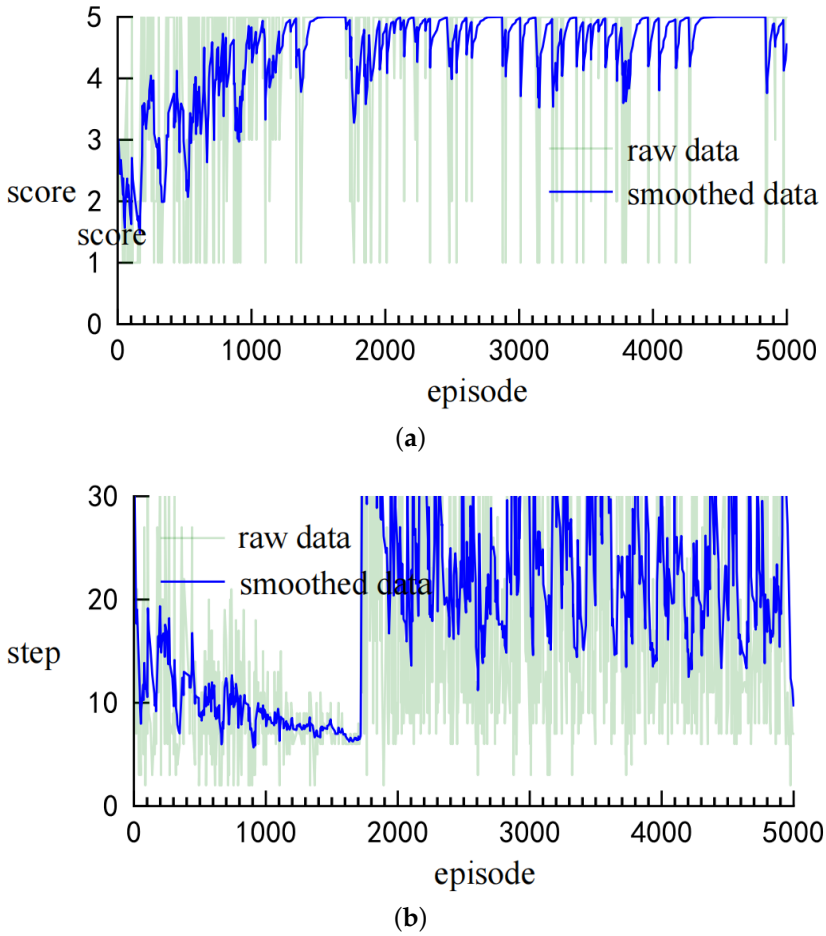


Figure 10. Jitter of the host intelligence without the invalid action discrimination mechanism. (**a**) Score of host3; (**b**) step of host3.

Table 2 presents the convergence steps and the jitter rate for penetration testing using the optimistic estimation mechanism and the invalid action discrimination mechanism. Experiment (1) which is shown in Figure 9, does not incorporate the optimistic estimation mechanism and fails to achieve convergence in training. Experiment (2) shown in Figure 10, lacks the invalid action discrimination mechanism, it exhibits both a high number of convergence steps and a high jitter rate. In contrast, the optimized Experiment (3) shown in Figure 8 is not only more efficient but also more stable.

Table 2. The network intelligence training algorithm performance.

Experiment	Convergence Steps with the Raw Data	Jitter Rate with the Raw Data	Convergence Steps with the Smoothed Data	Jitter Rate with the Smoothed Data
(1)	crash	crash	crash	crash
(2)	1700	0.43	1700	0.42
(3)	400	0.05	400	0.05

5.2. Experimental Validation of the Hierarchical Framework

The network testing scenario depicted in Figure 1 is designed to validate the effectiveness of the hierarchical architecture, which integrates network intelligence and host intelligence. The neural network parameters and training parameters are shown in Table 3.

Figure 11 illustrates the training curves of the network intelligence, with Figure 11a showing the scores for the intelligence and Figure 11b displaying the training steps. Leveraging pre-existing experience, the intelligence achieves its goals steadily after fewer than 200 exploration episodes and converges to the optimal path by approximately the 400th episode. The experimental results demonstrate that the network intelligence can effectively coordinate with the host intelligence to successfully perform penetration tasks.

Table 3. The parameters of the network intelligence.

Parameter	Set Value
Q-target network and Q-estimation network	input layer: obs_dim interlayer: 256 output layer: actions_dim
gamma	0.9
batchsize	32
epsilon	1
end epsilon	0.05
Q-target network update interval	200
learning rate	0.01

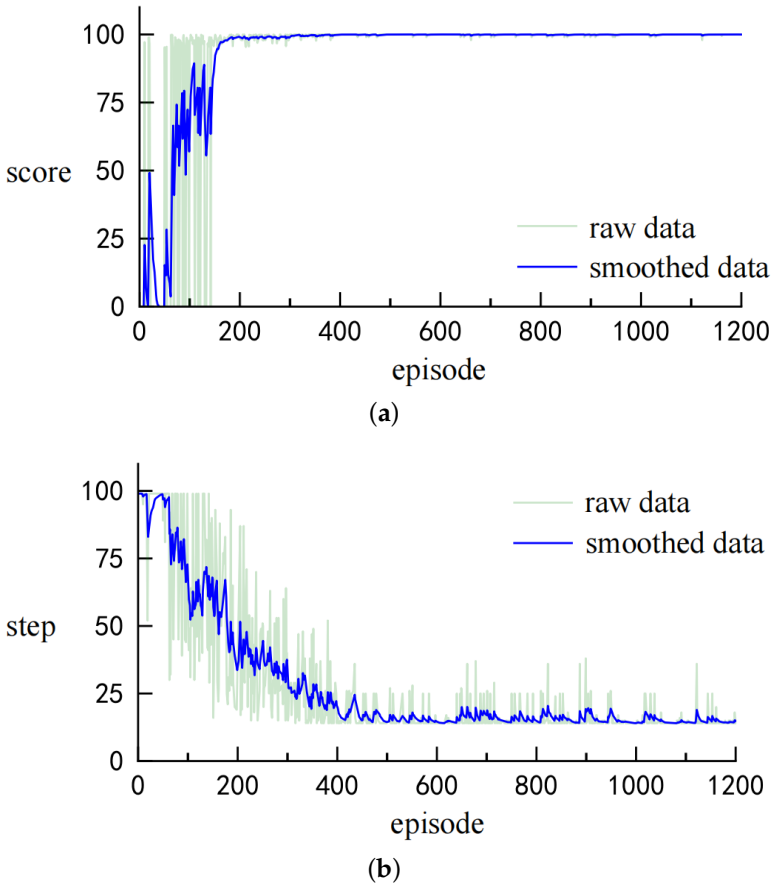


Figure 11. Training curves of the network intelligence. (**a**) Score of the network intelligence; (**b**) step of the network intelligence.

The time/fps reflects the intelligence's interaction speed with the environment and the frames updated per second corresponding to the number of steps. In the same network scenario, the GNN algorithm exhibits a decline in program execution speed as the number of steps increases, as shown in Figure 12. However, with the DQN algorithm used in this study, as the number of cycles increases, both the number of steps and the duration in each iteration decrease, resulting in higher efficiency.

The A3C algorithm is well suited for predefined security scenarios, requiring fewer actions than the DQN when conducting penetration testing [38]. However, the DQN offers ease of implementation and greater flexibility in some cases. It approximates the Q-function using a single deep neural network and adjusts hyperparameters to adapt to different environments and tasks. Additionally, the DQN utilizes an experience replay mechanism to store and reuse past experiences, which helps in breaking correlations between samples. Regular updates to the target network's weights further reduce fluctuations in target values, enhancing learning stability.

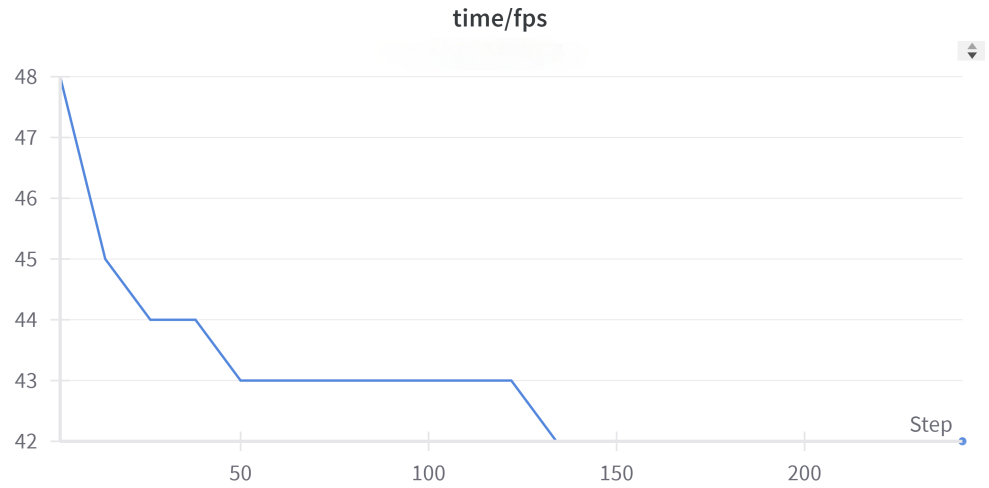
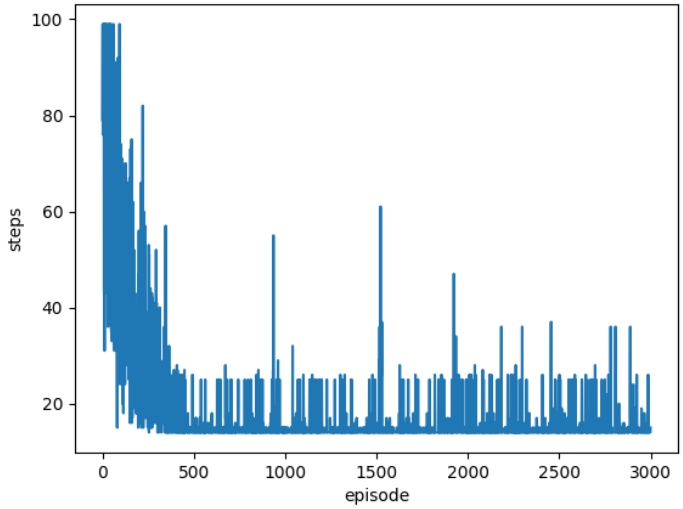


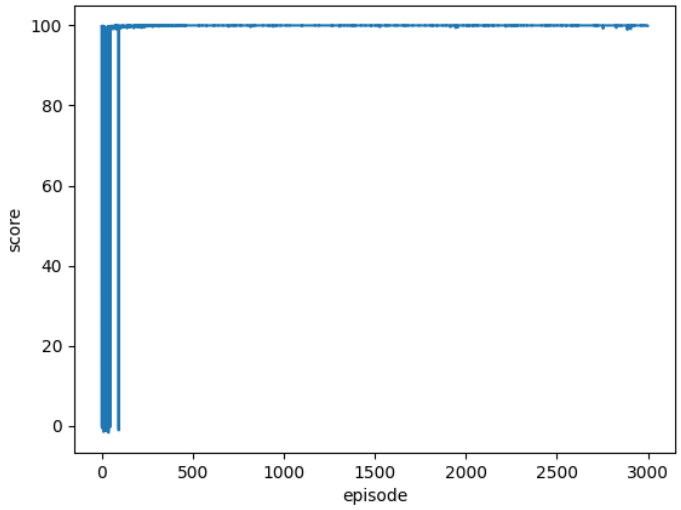
Figure 12. Interaction efficiency using the GNN algorithm.

Figures 13 and 14 show the results of steps and scores through 3000 iterations, with the number of hosts being increased to 13 and 17, respectively. As illustrated in the figures, the number of steps consistently decreases, reaching a minimum of 14 steps by the 255th episode. Meanwhile, the score stabilizes at 100 by the 139th episode. Importantly, the minimum steps and final scores remain unaffected by the increase in the number of hosts, demonstrating the stability of the penetration testing model proposed in this paper.



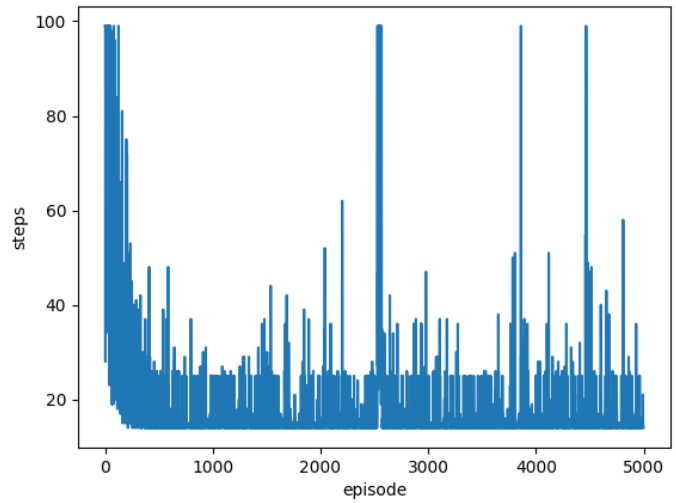
(**a**)

Figure 13. *Cont.*

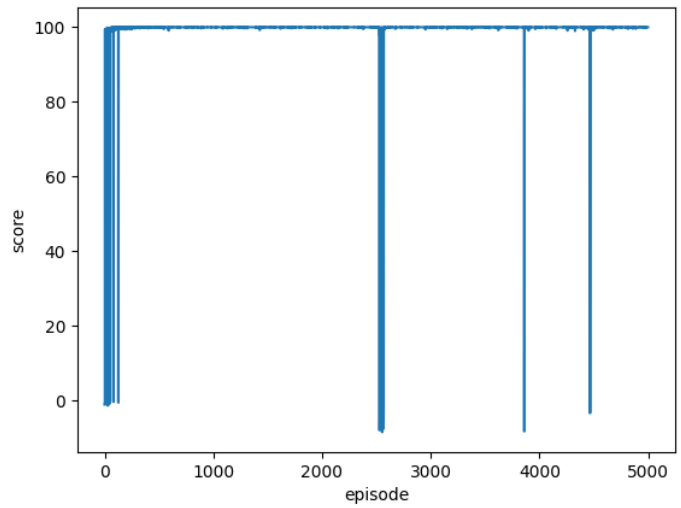


(**b**)

Figure 13. Training curves of 13 hosts. (**a**) Step of 13 hosts; (**b**) score of 13 hosts.



(**a**)



(**b**)

Figure 14. Training curves of 17 hosts. (**a**) Step of 17 hosts; (**b**) score of 17 hosts.

6. Conclusions

This paper presents a hierarchical reinforcement learning framework tailored for penetration testing scenarios. This framework integrates the action-masking mechanism with the network intelligence to efficiently manage unenabled actions. It also incorporates the active return action in the host intelligence to optimize penetration across hosts with varying depths. Furthermore, the invalid action discrimination mechanism is utilized to stabilize the training, while the optimistic estimation mechanism allows the host intelligence to adapt to multiple penetration strategies. In the hierarchical reinforcement learning framework, a different intelligence concentrates on their specific operations, thereby narrowing the invalid action space. It dynamically and efficiently adjusts strategies according to the requirements of various attack stages. This framework adapts to different penetration depths, offering valuable insights for advancing automated penetration testing technology. Additionally, the hierarchical network architecture is capable of uncovering complex attack chains and manipulations within IC, while also addressing security threats across various levels of IoT device management.Future research will focus on joint training strategies for dual-intelligence systems and the application of the invalid action mechanism in strategic reinforcement learning algorithms.

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

Funding: This research was funded by the National Natural Science Foundation of China (NSFC) (62272129), the Key Research and Development Program of Shandong Province (Competitive Innovation Platform) (2023CXPT065), the Small and Medium-sized Enterprise Capacity Re-improvement Project of Shandong Province (2022TSGC2459).

Data Availability Statement: The data presented in this study are available on reasonable request from the corresponding author.

Conflicts of Interest: Author Hongri Liu was employed by the company Weihai Cyberguard Technologies Co., Ltd. Author Hongmei Liu was employed by the company China Mobile Group Shandong Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

APT	Automated penetration testing
DQN	Deep Q-network
NIG-AP	Network information gain based automated attack planning
CRLA	Cascaded reinforcement learning agents
IAPTF	Intelligent automated penetration testing framework
H-DQN	Hierarchical deep Q-network
DAA	Double agent architecture
iPathD	Intelligent attack path discovery
OT network	Operational technology network
IT network	Information technology network
ICS	Industrial control system
TD error	Temporal-difference error
MDP	Markov decision process

References

1. Moscovich, N.; Bitton, R.; Mallah, Y.; Inokuchi, M.; Yagyu, T.; Kalech, M.; Elovici, Y.; Shabtai, A. Autosploit: A fully automated framework for evaluating the exploitability of security vulnerabilities. *arXiv* **2020**, arXiv:2007.00059.
2. Stefinko, Y.; Piskozub, A.; Banakh, R. Manual and automated penetration testing. Benefits and drawbacks. Modern tendency. In Proceedings of the 2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET), Lviv, Ukraine, 23–26 February 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 488–491.
3. Jijun, L.; Liusheng, H.; Shufeng, W. An Attack Tree Approach for Network Intrusion Modeling. *Comput. Eng. Appl.* **2003**, *27*, 160–163.
4. Tidwell, T.; Larson, R.; Fitch, K.; Hale, J. Modeling internet attacks. In Proceedings of the 2001 IEEE Workshop on Information Assurance and Security, United States Military Academy West Point, West Point, NY, USA, 5–6 June 2001; Volume 59.
5. Ammann, P.; Wijesekera, D.; Kaushik, S. Scalable, graph-based network vulnerability analysis. In Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington, DC, USA, 18–22 November 2002; pp. 217–224.
6. Greenwald, L.; Shanley, R. Automated planning for remote penetration testing. In Proceedings of the MILCOM 2009-2009 IEEE Military Communications Conference, Boston, MA, USA, 18–21 October 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 1–7.
7. Barto, A.G.; Mahadevan, S. Recent advances in hierarchical reinforcement learning. *Discret. Event Dyn. Syst.* **2003**, *13*, 341–379. [CrossRef]
8. Dayan, P.; Hinton, G.E. Feudal reinforcement learning. *Adv. Neural Inf. Process. Syst.* **1992**, *5*, 1–8.
9. Singh, S.P. Transfer of learning by composing solutions of elemental sequential tasks. *Mach. Learn.* **1992**, *8*, 323–339. [CrossRef]
10. Takahashi, Y.; Asada, M. Multi-controller fusion in multi-layered reinforcement learning. In Proceedings of the Conference Documentation International Conference on Multisensor Fusion and Integration for Intelligent Systems. MFI 2001 (Cat. No. 01TH8590), Baden-Baden, Germany, 20–22 August 2001; IEEE: Piscataway, NJ, USA, 2001; pp. 7–12.
11. Zeng, J.; Wu, S.; Chen, Y.; Zeng, R.; Wu, C. Survey of attack graph analysis methods from the perspective of data and knowledge processing. *Secur. Commun. Netw.* **2019**, *2019*, 2031063. [CrossRef]
12. Hu, Z.; Beuran, R.; Tan, Y. Automated penetration testing using deep reinforcement learning. In Proceedings of the 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Genoa, Italy, 7–11 September 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 2–10.
13. Chowdhary, A.; Huang, D.; Mahendran, J.S.; Romo, D.; Deng, Y.; Sabur, A. Autonomous security analysis and penetration testing. In Proceedings of the 2020 16th International Conference on Mobility, Sensing and Networking (MSN), Tokyo, Japan, 17–19 December 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 508–515.
14. Gangupantulu, R.; Cody, T.; Park, P.; Rahman, A.; Eisenbeiser, L.; Radke, D.; Clark, R.; Redino, C. Using cyber terrain in reinforcement learning for penetration testing. In Proceedings of the 2022 IEEE International Conference on Omni-Layer Intelligent Systems (COINS), Barcelona, Spain, 1–3 August 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1–8.
15. Takaesu, I. Deepexploit: Fully automatic penetration test tool using machine learning. *Blackhat Eur.* **2018**. Available online: https://github.com/13o-bbr-bbq/machine_learning_security/tree/master/DeepExploit (accessed on 12 August 2024).
16. Maeda, R.; Mimura, M. Automating post-exploitation with deep reinforcement learning. *Comput. Secur.* **2021**, *100*, 102108. [CrossRef]
17. Kujanpää, K.; Victor, W.; Ilin, A. Automating privilege escalation with deep reinforcement learning. In Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security, Virtual, 15 November 2021; pp. 157–168.
18. Zennaro, F.M.; Erdődi, L. Modelling penetration testing with reinforcement learning using capture-the-flag challenges: Trade-offs between model-free learning and a priori knowledge. *IET Inf. Secur.* **2023**, *17*, 441–457. [CrossRef]
19. Zhao, H.; Jiao, J. Recommendation model of penetration path based on reinforcement learning. *J. Comput. Appl.* **2022**, *42*, 1689.
20. Zhou, T.Y.; Zang, Y.C.; Zhu, J.H.; Wang, Q.X. NIG-AP: A new method for automated penetration testing. *Front. Inf. Technol. Electron. Eng.* **2019**, *20*, 1277–1288. [CrossRef]
21. Zhou, S.; Liu, J.; Hou, D.; Zhong, X.; Zhang, Y. Autonomous penetration testing based on improved deep q-network. *Appl. Sci.* **2021**, *11*, 8823. [CrossRef]
22. Wang, P.; Liu, J.; Zhong, X.; Yang, G.; Zhou, S.; Zhang, Y. DUSC-DQN: An Improved Deep Q-Network for Intelligent Penetration Testing Path Design. In Proceedings of the 2022 7th International Conference on Computer and Communication Systems (ICCCS), Wuhan, China, 22–25 April 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 476–480.
23. Tran, K.; Standen, M.; Kim, J.; Bowman, D.; Richer, T.; Akella, A.; Lin, C.T. Cascaded reinforcement learning agents for large action spaces in autonomous penetration testing. *Appl. Sci.* **2022**, *12*, 11265. [CrossRef]
24. Nguyen, H.V.; Nguyen, H.N.; Uehara, T. Multiple level action embedding for penetration testing. In Proceedings of the 4th International Conference on Future Networks and Distributed Systems, St. Petersburg, Russia, 26–27 November 2020; pp. 1–9.
25. Zhang, Y.; Liu, J.; Zhou, S.; Hou, D.; Zhong, X.; Lu, C. Improved deep recurrent q-network of POMDPs for automated penetration testing. *Appl. Sci.* **2022**, *12*, 10339. [CrossRef]
26. Zhou, W.J.; Yu, Y. Summarize of hierarchical reinforcement learning. *CAAI Trans. Intell. Syst.* **2017**, *12*, 590–594.
27. Dietterich, T.G. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.* **2000**, *13*, 227–303. [CrossRef]

28. Ghanem, M.C.; Chen, T.M.; Nepomuceno, E.G. Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks. *J. Intell. Inf. Syst.* **2023**, *60*, 281–303. [CrossRef]
29. Kulkarni, T.D.; Narasimhan, K.; Saeedi, A.; Tenenbaum, J. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Adv. Neural Inf. Process. Syst.* **2016**, *29*, 3675–3683.
30. Vezhnevets, A.S.; Osindero, S.; Schaul, T.; Heess, N.; Jaderberg, M.; Silver, D.; Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. In Proceedings of the International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; pp. 3540–3549.
31. Nguyen, H.V.; Teerakanok, S.; Inomata, A.; Uehara, T. The Proposal of Double Agent Architecture using Actor-critic Algorithm for Penetration Testing. In Proceedings of the ICISSP, Virtual, 11–13 February 2021; pp. 440–449.
32. Zeng, Q.; Zhang, G.; Xing, C.; Song, L. Intelligent Attack Path Discovery Based on Hierarchical Reinforcement Learning. *Comput. Sci.* **2023**, *50*, 308–316.
33. Sutton, R.S.; Barto, A.G. Reinforcement learning: An introduction. *Robotica* **1999**, *17*, 229–235. [CrossRef]
34. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
35. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30.
36. Huang, S.; Ontañón, S. A closer look at invalid action masking in policy gradient algorithms. *arXiv* **2020**, arXiv:2006.14171. [CrossRef]
37. Standen, M.; Lucas, M.; Bowman, D.; Richer, T.J.; Kim, J.; Marriott, D. Cyborg: A gym for the development of autonomous cyber agents. *arXiv* **2021**, arXiv:2108.09118.
38. Becker, N.; Reti, D.; Ntagiou, E.V.; Wallum, M.; Schotten, H.D. Evaluation of Reinforcement Learning for Autonomous Penetration Testing using A3C, Q-learning and DQN. *arXiv* **2024**, arXiv:2407.15656.