A Path Discovery Method of Penetration Testing Based on SAC Reinforcement Learning*

Yimeng Liu[1,*,†], Xiaojian Liu[2,†] and Xuejun Yu[3,†]

[1]Beijing University of Technology, Beijing, China
[2]Beijing University of Technology, Beijing, China
[3]Beijing University of Technology, Beijing, China

Abstract

As the complexity and scale of systems continue to increase, enterprises are placing ever higher demands on system security, making comprehensive security analysis particularly important. Among the various methods, penetration testing is regarded as one of the most direct means for assessing security. In order to explore the resistance (to attacks) standard within the SQuaRE series standards concerning product quality, this paper proposes a penetration path discovery method based on an improved deep reinforcement learning algorithm, Adaptive Soft Actor-Critic (ASAC). This method involves modeling the penetration process and quantifying the penetration benefits, and it leverages the Soft Actor-Critic (SAC) algorithm, which is suitable for complex state spaces, to construct an intelligent penetration agent. The goal is to solve for the optimal penetration path, thereby enabling the evaluation of a system's resistance to attacks during system validation.Experimental results demonstrate that the attack effectiveness of the proposed ASAC algorithm surpasses that of commonly used reinforcement learning algorithms such as Q-learning and DQN. It can quickly identify the most critical penetration paths in a network and maintain high performance across different network environments. This approach provides effective theoretical and technical support for the comprehensive assessment of the robustness of the cybersecurity of the system.

Keywords

Deep Reinforcement Learning, Penetration test, Attack path discovery

1. Introduction

As business scales grow, many enterprises choose to store data across multiple physical devices (such as servers) to enhance access speeds, with unified management and access via networks. Attackers may infiltrate the data storage local area network through boundary nodes, thereby stealing sensitive information and causing significant losses to the enterprise. It is crucial for enterprises to periodically conduct penetration testing based on the state of their network environment, to assess the system's risk and robustness. Penetration testing is a black-box security testing method, in which the tester adopts the mindset and technical means of an attacker to detect vulnerabilities in business system targets. It helps enterprises uncover hidden security flaws and vulnerabilities in normal business processes, breach the system's security defenses, and deeply assess the potential impacts of vulnerabilities.

In real-world enterprise internal networks and sensitive core networks, initial stages of penetration testing may not provide valuable information due to the lack of direct insight into the system. The Markov process (Markov Process) is a type of stochastic process characterized by the property that future states depend only on the current state, not on past states. This paper formalizes the penetration testing process as a Markov Decision Process (MDP), utilizing network information to build a reward function that guides the agent to discover hidden attack paths from the attacker's perspective. This method does not require prior validation of network structures or software configurations and can autonomously discover attack paths, extracting essential penetration testing information. Furthermore, reinforcement learning is employed as the agent's learning strategy, enabling it to interact with the environment to maximize its reward. Penetration testing can also be viewed as a dynamic decision-making process based on the current environmental state. Given the similarity between penetration testing and reinforcement learning mechanisms, reinforcement learning is well-suited to describe penetration testing in unknown environments.

This paper proposes an improved reinforcement learning algorithm, ASAC, for penetration path planning in unknown environments. By improving search strategies and action selection policies, the algorithm enhances the planning efficiency.

Additionally, the proposed method aligns with the ISO/IEC 25000 series standards, specifically the SQuaRE (Software Quality Requirements and Evaluation) framework, particularly the security resistance sub-characteristics. According to the SQuaRE standard, security, as an important aspect of software quality, requires that the system be able to resist attacks and ensure the confidentiality, integrity and availability of data. Our proposed penetration testing method, by optimizing the attack path discovery, can evaluate the system's resistance and defense capability according to the calculation time of the model and the success rate of multiple simulated attacks. In the ISO/IEC 25010 standard corresponding security features such as confidentiality, integrity, availability and resistance have been effectively improved.

The main contributions of this paper are as follows:

- We combines complex multi-step attack characteristics to conduct penetration test modeling, quantifies action gains and network environment vulnerability losses, and more accurately simulates the process of attackers gradually breaking through various firewalls and gateways, providing accurate reward signals for the generation of multi-step and multi-host attack paths.
- SAC reinforcement learning algorithm suitable for

6th International Workshop on Experience with SQuaRE family and its Future Direction,3rd December 2024,Chongqing, China(Hybrid)

* You can use this document as the template for preparing your publication. We recommend using the latest version of the ceurart style.

*Corresponding author.
†These authors contributed equally.
lymon1004@163.com (Y. Liu); liuxj@bjut.edu.cn (X. Liu); 759301040@qq.com (X. Yu)
0009-0009-6580-9840 (Y. Liu); 0000-0002-0666-4102 (X. Liu)

complex discrete action or even continuous action space is introduced into the penetration testing process, and the improved intelligent penetration testing agent (ASAC) can cope with the complex and changeable network environment and the exponential growth of the action space.

2. Related work

Traditional penetration testing (PT) relies on manual methods, which become impractical as the systems grow in size and complexity. By simulating 20 attack strategies of real attackers, automated penetration testing technology uses various algorithm models to automate the penetration of 21 target networks, significantly reducing test cost and improving penetration efficiency [1] [2]. Automated penetration testing is an important application of artificial intelligence technology in the field of network security [3]. Zang et al. [4] summarized the current research progress of attack path discovery in automated penetration testing, and proposed future research directions. Modeling the attack environment through Markov process can well help us describe and calculate the penetration test path. Literature [5] modeled the environment as Markov decision process diagram based on the attack graph, and used the value iterative algorithm to find the optimal penetration path. This approach can help penetration testers find the most effective attack path, thereby improving the efficiency and success rate of penetration testing. Literature [6] proposes an automatic attack planning algorithm (NIG-AP) based on network information gain, which formalizes penetration testing into a Markov decision process, finds potential attack paths from the attacker's perspective, and uses network information to optimize attack strategies. Zhou et al. [7] proposed an attack path planning algorithm based on network information gain, which formalized penetration testing into a Markov decision process, used network information to obtain rewards, and guided agents to choose the best response actions. The method based on reinforcement learning, which can simulate the uncertainty of offense and defense in the real world by designing the probability of success of action execution, is an important research direction in this field. Schwartz et al. [8] designed a lightweight network attack simulator NASim, which provides a benchmark platform for network attack defense simulation test. They verify the effectiveness of basic reinforcement learning algorithms, such as Deep Q-learning Network(DQN), in the application of penetration path discovery. In order to improve the convergence speed of DQN algorithm in path discovery problems, Zhou Shicheng et al. [9] proposed an improved reinforcement learning algorithm, noise-double-dueling DQN. Hoang Viet Ngueyen et al. [10] introduced an A2C algorithm with dual agent architecture, which is responsible for path discovery and host utilization respectively. Zeng Qingwei et al. [11] suggested using hierarchical reinforcement learning algorithm to solve the problem of path discovery and host utilization being handled separately. In summary, current research on the discovery of intelligent penetration test paths is still in the preliminary stage. The advantage of MDP model is that it can model the uncertainty in the process of penetration testing well, but it brings the increase of computational complexity, which is difficult to apply to large-scale network scenarios. The method based on reinforcement learning is only experimentally verified in simple network scenarios, and there is still much room for improvement in the convergence speed and scalability of the algorithm.Based on the above research, we further study penetration testing in complex scale scenarios. In this paper, the infiltration test attack path is modeled from the perspective of multi-step attack, and it is combined with MDP. On this basis, an intelligent decision method for complex computing network attacks is proposed.

3. Methodology

3.1. Markov Decision Model

Figure 1 shows a typical penetration testing scenario, which we use as a case study to illustrate the issues discussed in this article. The target network consists of the extranet, DMZ, and intranet subnets. The subnets contain user nodes and data nodes (sensitive nodes). The function of the intelligent penetration agent is to determine the attack target of each step according to the state observation to reach the final sensitive node and plan the optimal scheme.

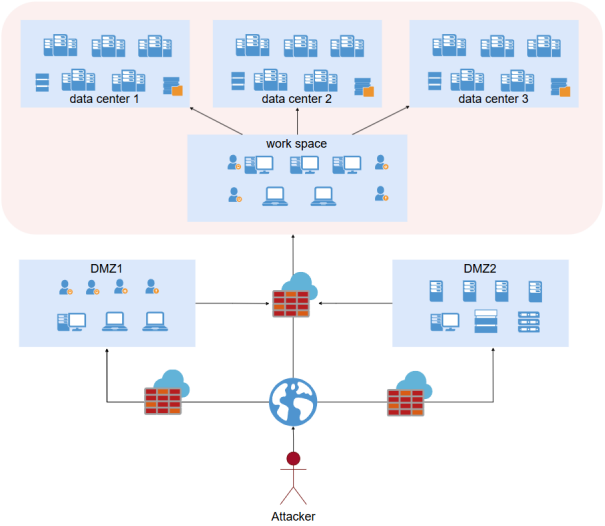

Figure 1: Example of Penetration Testing

The Markov Decision Process (MDP) is commonly modeled as a quadruple $\langle S, A, R, P \rangle$, where:

- S represents the set of penetration states observed by the agent, such as network topology, host information, and vulnerability details. These states correspond to various stages in the penetration testing process, which typically proceeds in the following sequence:
  - **a) Information Gathering**: The initial phase, where information regarding the network, systems, and potential vulnerabilities is collected.
  - **b) Vulnerability Assessment**: The process of identifying and evaluating weaknesses or vulnerabilities within the network and systems.
  - **c) Exploitation**: The stage where discovered vulnerabilities are exploited to gain unauthorized access to systems.

  - : **d) Privilege Escalation**The process of elevating access privileges once entry has been gained, allowing deeper penetration into the network.
  - **e) Maintaining Access**: Ensuring persistent access to the compromised system.
  - **f) Reporting**: The final phase, in which findings are documented, including discovered vulnerabilities and the level of access achieved.
- A denotes the set of possible attack actions, corresponding to the aforementioned stages. These actions include network scanning, host scanning, vulnerability exploitation, privilege escalation, and others available to the agent.
- R is the reward function $R(s)$, which assigns a reward based on the different penetration states. For example, gaining the highest level of permission on a sensitive host might yield a reward of 100, while breaching an Intranet subnet host could result in a reward of 1, and no reward is given for breaching a DMZ host.
- P represents the state transition function $P(s, a, s') = Pr(s'|s, a)$, which defines the probability of transitioning from one state to another after performing a given action. This is typically associated with the success rate of the attack actions.

The objective of the MDP is to select the optimal policy $a_t = \pi(s_t)$ that maximizes the long-term cumulative reward $G(s_0)$ for the current state, as expressed in equation (1).

$$G(s_0) = E\left\{\sum_{t=0}^{T-1} \gamma^t R(s_{t+1}) P[s_t, \pi(s_t), s_{t+1}]\right\} \quad (1)$$

where $\gamma \in (0, 1)$ denotes the discount factor, used to balance the importance of current rewards against future rewards.

3.2. Attack cost quantification model

Designing a quantitative attack cost model can comprehensively consider the investment and return of attack behavior from multiple dimensions, so as to provide a more objective basis for network security evaluation. For the MDP process discussed above, the penetration test adopts multiple attack modes to penetrate the network. Assuming that the threat of a certain type of attack is X, the launching cost of such an attack is A, the attack cost of node i is N, and the attack value is V, then the attack cost of a certain node j is attacked. Suppose that on the optimal attack path, the state of node i relative to the attacker is S,After the attack fails, the network will detect the attack, thus increasing the attack cost.LC(limit cost) a given cost, it is assumed that the resources held by the attacker can consume LC at most, which can help the intelligent agent accelerate the convergence speed during the network search attack.

$$C = \min(C) = \sum_{i<j} (a_i + n_i) \cdot s_i \quad (2)$$

it satisfies inequality as (3):

$$C <= LC \quad (3)$$

Damage to the network as (4):

$$W = \max(W) = \sum_{i<j} (x_i + v_i) \cdot p_i \quad (4)$$

So the attack reward function (5) in the current state

$$R = W - C = \sum_{i<j} \{(x_i + v_i) \cdot p_i - (a_i + n_i) \cdot s_i\} \quad (5)$$

P is the breach probability of a node, and the attack threat degree is related to the corresponding service vulnerability scores of different nodes in different networks.

3.3. Improved Soft Actor-Critic Algorithm

Reinforcement learning algorithms can be categorized into three main types: value-based, policy-based, and actor-critic-based. SAC is an off-policy reinforcement learning algorithm that combines maximum entropy learning with the actor-critic framework. It learns a policy network to select optimal penetration actions while estimating state and action values, thereby maximizing the policy's entropy to encourage exploration and diversification in attack path selection. Currently, SAC is an efficient model-free reinforcement learning algorithm capable of learning stochastic policies, achieving state-of-the-art results in many standard environments. Figure 2 shows the architecture of the improved Attack Soft Actor-Critic(labelled as ASAC) algorithm. The ASAC algorithm consists of a policy network and four value function networks. The policy network acts as the actor, outputting attack actions toward the environment, while the four value networks evaluate the policy network. Since it is a model-free algorithm, SAC uses an experience replay buffer to store all actions and environmental feedback data, and updates the networks through random sampling. Given that actions in reinforcement learning are often highly correlated, this approach allows the neural network to achieve more effective training.Different from other RL algorithms, in order to encourage exploration, the concept of entropy is added in SAC algorithm, and entropy regularization increases the exploration degree of reinforcement learning algorithm. The greater α is, the stronger the exploration is, which helps to accelerate the subsequent strategy learning and reduce the possibility of the strategy falling into poor local optimal.

3.3.1. Policy network design

It can be seen that the input of the policy network Actor is the network status s_t, the output is $Pi(ai|St)$ action policy, and the update loss expression of its neural network is :

$$\text{MSELoss} = -\frac{1}{|\mathcal{B}|} \sum_{(s_t, a_t, r_{t+1}, s_{t+1}) \in \mathcal{B}} E_{a'_t \sim \pi(\cdot|s_t;\theta)} \left[q_0(s_t, a'_t) - \alpha \ln \pi(a'_t|s_t;\theta)\right] \quad (6)$$

$$\mathbb{E}_{a'_t \sim \pi(\cdot|s_t;\theta)} \left[q_0\left(s_t, a'_t; w^{(0)}\right) - \alpha \ln \pi(a'_t|s_t;\theta)\right] = \sum_{a'_t \in \mathcal{A}(s_t)} \pi(a_t|s_t;\theta) \left[q_0\left(s_t, a'_t; w^{(0)}\right) - \alpha \ln \pi(a'_t|s_t;\theta)\right] \quad (7)$$

You can observe the Equation (7) that the three elements in the equation — $\pi(a_t|s_t;\theta)$, $q_0\left(s_t, a'_t; w^{(0)}\right)$,

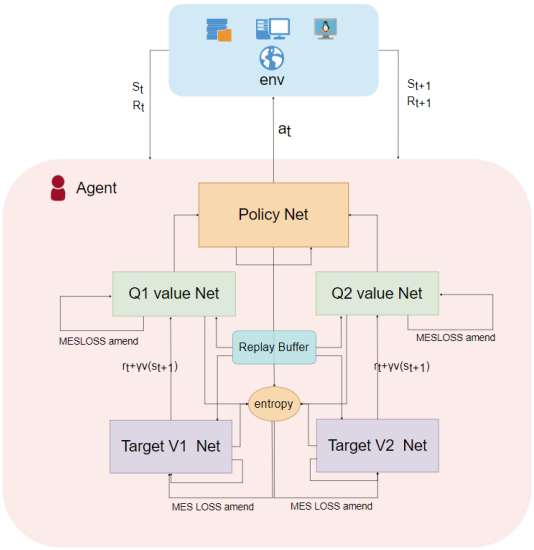

Figure 2: Architecture of ASAC

and $\ln \pi\left(a_t' | s_t ; \theta\right)$ — are fully aligned with the Loss function depicted in the graph. It's important to note that $q_0\left(s_t, a_t' ; w^{(0)}\right)$ can be replaced by $q_1\left(s_t, a_t' ; w^{(1)}\right)$, since both Q-critic networks function equivalently.Based on the idea of Double DQN, SAC uses two Critic networks, but each time a Critic network is used, it picks a network with a small value, thereby alleviating the problem of overestimation.

The symbol $\mathcal{B}$ represents the experience buffer, meaning that when calculating the Loss, you need to take the average of the samples drawn from the buffer. This ensures that the expected average meaningfully represents the sample's overall outcome.

Here, α is the entropy coefficient, which controls the importance of the entropy term $\ln \pi\left(a_{t+1} | s_t ; \theta\right)$, and its significance increases as α increases.

3.3.2. Q Critic network design

Based on the optimal Bellman equation, we use $U_t^{(q)} = r_t + \gamma V(s_{t+1})$ as the true value estimate for the state s_t, while the $q_i(s_t, a_t)$ value (where $i = 0, 1$) is used as the predicted value estimate for state s_t with the actual action a_t. Finally, the MSE loss is used as the loss function to train the neural networks Q_0 and Q_1.

Note that using MSE loss implies taking the average of the data sampled from the experience buffer (denoted as $\mathcal{B}$) across a batch, as follows:

$$\text{Loss} = \frac{1}{|\mathcal{B}|} \sum_{(s_t, a_t, r_t, s_{t+1}) \in \mathcal{B}} \left[q_i\left(s_t, a_t ; w^{(i)}\right) - U_t^{(q)} \right]^2$$

3.3.3. V Critic network design

Using the following equation with entropy for state value estimation, the V critic network outputs the true value:

$$U_t^{(v)} = \mathbb{E}_{a_t' \sim \pi(\cdot | s_t ; \theta)} \left[\min_{i=0,1} q_i\left(s_t, a_t' ; w^{(i)}\right) - \alpha \ln \pi\left(a_t' | s_t ; \theta\right) \right]$$

which can be written as:

$$U_t^{(v)} = \sum_{a_t' \in \mathcal{A}(s_t)} \pi\left(a_t' | s_t ; \theta\right) \left[\min_{i=0,1} q_i\left(s_t, a_t' ; w^{(i)}\right) - \alpha \ln \pi\left(a_t' | s_t ; \theta\right) \right]$$

We observe that $\pi\left(a_t' | s_t ; \theta\right)$, $\min_{i=0,1} q_i\left(s_t, a_t' ; w^{(i)}\right)$,$\ln \pi\left(a_t' | s_t ; \theta\right)$ in these terms perfectly match the loss calculation described in Figure 2.

The output of the V critic network is used as a predicted value, and finally, MSE loss is applied as the loss function, training the V neural network.

3.3.4. Maximum entropy reinforcement learning

As we mentioned earlier,entropy represents a measure of the degree of randomness of a random variable. Specifically, if X is a random variable and its probability density function is p, then its entropy $H(X)$ is defined as $H(X) = \mathbb{E}_{x \sim p}[-\log p(x)]$.

3.3.5. Regional Experience Replay Buffer

In a large-scale network environment, direct use of a single experience playback pool may cause the experiences of different subnets or nodes to interfere with each other, thus reducing training efficiency. To solve this problem, the experience playback pool can be divided into multiple regional playback pools (each region corresponds to a subnet or node) to store and utilize the experience more targeted.

Divide the experience pool: The experience pool is divided into multiple subpools based on the network topology. Each subpool is dedicated to storing experiences from a particular subnet or node. This division ensures that the experience within each region is more homogeneous and avoids the mixing of different regional experiences.

Intra-zone sampling: At each policy update, SAC agents can sample experiences from the playback pool associated with the current zone of operation. For example, if the agent is currently in subnet A, it will preferentially sample from subnet A's experience pool. This targeted sampling method can improve the learning efficiency of the model, because the experience of sampling is more relevant and can optimize the strategy within the region faster.

Cross-region sampling: In a partial update step, a pool of experience from different regions can be sampled to enhance the model's adaptability to the global network. This approach balances the relationship between regional focus and global exploration, so that the model can not only focus on the strategy optimization of a specific region, but also understand the dynamics of other regions and enhance the generalization ability.

Mitigate non-stationarity issues: In complex network environments, the dynamics of subnets may not be synchronized. For example, some subnets may change frequently, while others are relatively stable. By storing the experience pool in different regions, the negative impact of non-stationarity on the model can be prevented and the stability of training can be effectively maintained.

3.3.6. Prioritized Experience Replay

In reinforcement learning, not all experiences have the same impact on policy improvement. Prioritized experience replay selects experiences that contribute more significantly

to policy improvement, helping the SAC algorithm make efficient use of critical experiences and accelerate learning progress.

1. **Priority Calculation**: Each experience is assigned a priority, typically calculated based on the Temporal Difference (TD) error:

$$\delta = \left| r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right|$$

The greater the TD error, the higher the potential for policy improvement, and therefore, the higher the priority.

2. **Experience Sampling**: During sampling, experiences with higher priority are more likely to be chosen. This can be achieved by probability sampling, where the sampling probability $P(i)$ is set proportional to the priority level p_i. For example, the sampling probability $P(i)$ can be defined as:

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

where p_i is the priority of experience i, and α controls the level of prioritization. When $\alpha = 1$, sampling is fully prioritized by priority; when $\alpha = 0$, sampling is uniform.

3. **Importance Sampling Weight Correction**: To correct for sampling bias, higher-priority experiences are given lower weights in the gradient calculation. The importance sampling weight $w(i)$ can be used to balance this bias:

$$w(i) = \left(\frac{1}{N \cdot P(i)} \right)^\beta$$

where N is the size of the experience buffer, and β is a parameter that adjusts the degree of importance-sampling correction. When $\beta = 1$, the correction is fully applied. Typically, β starts from 0 and gradually increases to avoid instability due to high weights early in training.

The flow of the improved ASAC(Attacker Soft Actor Critic) algorithm is as Algorithm 1.

Line 1-4: Initialize the Critic, Actor, and target network parameters, along with regional experience replay buffers and temperature parameter. Line 5: Start the loop for each episode, sampling the initial state and determining the region. Line 7-10: At each time step, sample an action, execute it, observe the outcome, and store the experience in the respective regional replay buffer. Line 11-19: Perform training steps by sampling a mini-batch with prioritized experience replay, calculating TD errors and priorities, updating Critic and Actor networks, and adjusting temperature. Line 20: Update the target networks with a soft update mechanism.

Algorithm 1 Framework of ASAC Algorithm

Initialize Critic network parameters ω_1, ω_2 and Actor network parameters θ
Initialize target network parameters ω_1^-, ω_2^-
Initialize regional experience replay buffers $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_N$
Initialize temperature parameter α
for each episode **do**
  Sample initial state s_1, determine the current region
  for each time step $t = 1 \to T$ **do**
    Sample action $a_t \sim \pi_\theta(\cdot|s_t)$ from the policy network
    Execute action a_t, observe reward r_t and next state s_{t+1}
    Store (s_t, a_t, r_t, s_{t+1}) in the corresponding regional replay buffer $\mathcal{R}_i$
    for training steps $k = 1 \to K$ **do**
      Sample a mini-batch $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^N$ from all regional replay buffers using prioritized sampling
      Calculate the TD error $\delta_i = r_i + \gamma \min_{j=1,2} Q_{\omega_j^-}(s_{i+1}, a_{i+1}) - Q_{\omega_j}(s_i, a_i)$
      Update priority for each sample based on δ_i and adjust sampling probabilities
      Calculate target value $y_i = r_i + \gamma \min_{j=1,2} Q_{\omega_j^-}(s_{i+1}, a_{i+1})$
      Update Critic networks: for $j = 1, 2$, minimize the loss function
      Update Actor network using reparameterization trick to sample action $\tilde{a}_i$
      Update temperature parameter α to minimize entropy objective
    end for
    Update target network parameters:

$$\omega_j^- \leftarrow \tau\omega_j + (1-\tau)\omega_j^-, \quad j = 1, 2$$

  end for
end for

4. Experiment and Discussion

4.1. network configuration

To simulate a realistic observation-based penetration process, we assume that both topology and host information must be obtained through scanning or feedback from attack actions. Therefore, the Information Gathering step is carried out through 4 steps: (1) subnet scan (subnet_scan) to discover all hosts in the subnet; (2) operating system scan (os_scan) to obtain the operating system type of the target host; (3) service scan (service_scan) to obtain the service type of the target host; (4) process scan (process_scan) to obtain information about processes on the target host. By performing the scan actions, the PT Agent can acquire the corresponding host information as an observed state. The VE actions and PE actions must be performed based on the specific requirements. In addition, a certain probability of success is set according to the Common Vulnerability Scoring System (CVSS) [12] to simulate the uncertainty of attacks in reality. By configuring different VE actions and PE actions, the PT agent is modelled with different attack capabilities, for example as shown in Table 1. The results of the attack actions are simulated by updating the compromised state information of the target host.

The network topology diagram of the simulation experiment in this paper is shown in Figure 3: A total of 6 subnets, each subnet is set between the firewall and access protocol, the yellow host is the sensitive host in the network, the attacker has mastered the two available nodes (1,0), (6,0) and 4 scanning means, VE, PE and other actions to finally obtain the sensitive node permissions and related sensitive

Table 1
Attack Actions

Type	Name	Prerequisites			Results	
		OS	Service	Process	Prob	Privileges
VE	E_SSH	Linux	SSH	—	0.9	User
	E_FTP	Windows	FTP	—	0.6	User
	E_HTTP	—	HTTP	—	0.9	User
PE	P_Tomcat	Linux	—	Tomcat	1	Root
	P_Daclsvc	Windows	—	Daclsvc	1	Root

Attack capabilities configuration in Figure 3

information, the vulnerability information of the network is shown in the Figure 3. The host configuration is shown in Table 2.

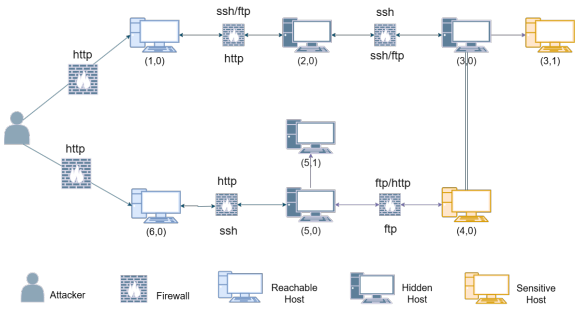

Figure 3: Network topology sample

Table 2
Host Configurations

Host	OS	Services	Processes
(1, 0)	Linux	http	—
(2, 0)	Linux	ssh, ftp	tomcat
(3, 0)	Windows	ftp	—
(3, 1)	Linux	ssh	—
(4, 0)	Windows	http	daclsvc
(5, 0)	Linux	ftp, ssh	—
(5, 1)	Windows	ftp	daclsvc
(6, 0)	Linux	http	tomcat

4.2. Tests and Analyses

To validate the performance of our model and decision method, we focus our experiments on answering the following three Research Questions (RQs):

4.2.1. RQ1: indicates whether the ASAC algorithm is feasible in the network environment after parameter tuning?

In this part, we first perform detailed parameter tuning for the ASAC algorithm, and a total of four groups of parameters are set, with detailed data shown in Table 3. The goal of parameter tuning is to find an optimal set of hyperparameters to ensure the stability and effectiveness of the algorithm in the network environment. We used a combination of grid search and random search to explore different combinations

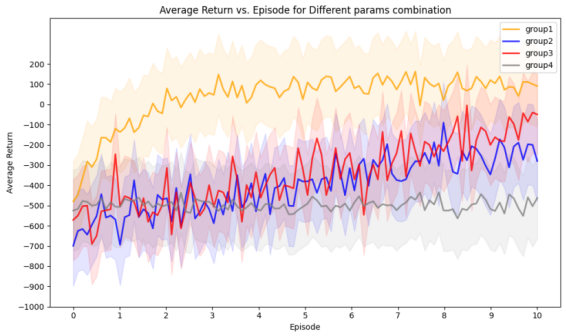

Figure 4: Different params combination Tests

of hyperparameters, including learning rate, discount factor, batch size, and so on. Through many experiments, we identify a set of optimal hyperparameters and verify them in the network environment.

Figure 4 shows the performance comparison of ASAC algorithm under different parameter Settings. The experimental results of each group are smoothed. The experimental results show that the proxy training obtained by group4 group parameters cannot achieve balance, and basically each training round cannot reach the optimal condition. But it doesn't converge yet. After multiple tuning, the optimal parameter group group1 is found. After parameter tuning, the ASAC algorithm shows good performance in the network environment and can converge in about 300 rounds. It can be seen that the optimized algorithm has a significant improvement in convergence speed and final performance.

4.2.2. RQ2: Compared with previous algorithms, does the improved ASAC algorithm have better performance in the network environment?

In order to verify the performance advantages of the improved ASAC algorithm in the network environment, we compare the ASAC algorithm with several classical network optimization algorithms, including Q-learning and DQN (Deep Q-Network), which have been studied in intelligent penetration testing.

Figure 4 shows the change of the cumulative reward value of each round of different algorithms with the number of training rounds in the network environment. The learning goal of the agent is to learn to use fewer steps to obtain the permissions of all sensitive hosts in the target network, so as to obtain the reward value, so the reward value can be used to measure the level of the agent's strategy. It can be found from the Figure 5 that in the initial stage of training,

Table 3
Hyperparameter Tuning Table Group

Hyperparameter	group1	group2	group3	group4
Actor Learning Rate (α)	1e-3	1e-3	1e-3	1e-4
Critic Learning Rate (α)	1e-3	1e-3	1e-3	1e-4
Discount Factor (γ)	0.9	0.95	0.95	0.95
Standard Batch Size	128	64	32	32
Prioritized Replay Batch Size	128	64	32	32
Target Update Rate (τ)	5e-3	5e-2	5e-2	1e-3
Replay Buffer Size	3e5	3e5	3e5	3e5
Prioritization Parameter (α)	0.8	0.8	0.8	0.8
Regional Buffer Size	5000	5000	5000	5000
Exploration Noise	0.05	0.05	0.1	0.1

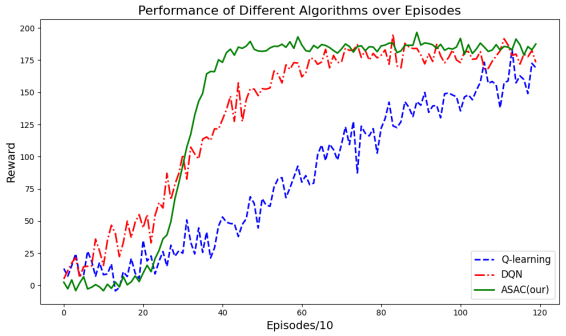

Figure 5: Different algorithms Tests

the reward value that the agent can obtain in each turn is small, but with the progress of training, the reward value keeps increasing, indicating that the agent gradually learns the strategy of obtaining the maximum reward value. In the early exploration process, the cumulative reward value of ASAC algorithm per round is significantly lower than that of DQN and Q-learning algorithm. This is because in the early exploration process of ASAC algorithm, the agent has no experience to rely on and can only use random strategies for exploration. It also wastes a lot of attack steps, but with the progress of training Q-learning can not get better results due to the explosion of Q table, and the convergence speed and effect of DQN are not as good as that of ASAC algorithm. Finally, both ASAC and DQN algorithms converge to a stable cumulative reward value within 600 rounds, and ASAC algorithm has the best performance and can converge to the optimal value within 300 rounds.

The experimental results show that the improved ASAC algorithm has obvious advantages in network environment. Including convergence speed, decision quality and resource utilization. ASAC algorithm outperforms other algorithms in all indexes, especially in the environment dealing with large-scale networks and high dynamic changes.

4.2.3. RQ3: Is the proposed ASAC scalable across different network sizes?

In order to verify the scalability of ASAC algorithm, we conducted experiments in different scale network environments. The experiment was carried out in a network simulation environment to simulate network topologies of different sizes and complexity. Table 4 shows five different scale network environment configurations used in the experiment.

The experimental networks range in size from 8 to 64 nodes and cover different scenarios from small Lans to large WAN networks. The main indicators we focus on are the score of ASAC algorithm under different network conditions, and the convergence speed.

Table 4
Network Environment Specifications

Env Name	Nodes	Subnets	Sensitive Nodes
Environment A	8	5	2*100
Environment B	8	6	2*100
Environment C	9	6	2*100
Environment D	32	8	5*100
Environment E	40	10	10*100

As can be seen from Figure 6, in the three heterogeneous networks A, B and C with the same sensitive host and slightly different summary points and subnets, the ASAC algorithm agent has good robustness, and its average cumulative reward value can converge to the optimal value within 400 rounds in the three similar scenarios.

As the number of subnets and hosts increases, the convergence rate of the algorithm slows down as the number of hosts and actions that the agent needs to attempt in each turn increases rapidly. Figure 7 shows the performance of ASAC algorithm under different network scales. It can be seen that the performance of ASAC algorithm decreases slightly with the increase of network scale, but it can still effectively deal with complex problems in large-scale network environment. In the simulated network environment of 40 hosts,ASAC algorithm can also converge to the optimal solution. The experimental results show that the ASAC algorithm maintains good performance when the network size increases gradually. In addition, ASAC algorithm also performs well in resource utilization, and can maintain high resource utilization under different network sizes. When dealing with complex network topology and high load, ASAC algorithm can make decisions effectively to ensure the stability and efficiency of the network.

5. Conclusion

Through the experimental analysis, we can draw the following conclusions:

These experimental results validate the effectiveness and

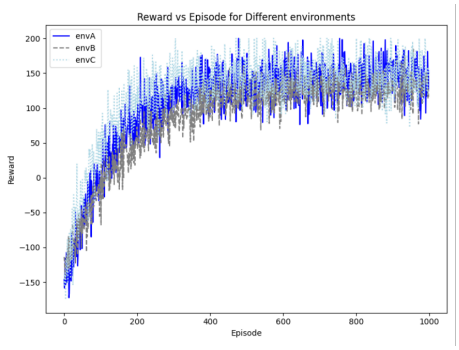

Figure 6: Reward in similar network environment

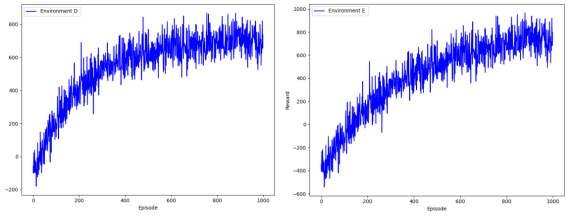

Figure 7: Reward in complex network environment

superiority of ASAC algorithm, provide an effective tool for network intelligent penetration attack agents, and also provide evaluation support for the security subfeature "resistance" defined in ISO/IEC 25000 SQuaRE product quality model. It provides strong support for the future application in the field of network optimization.

Quantitative analysis reveals that the ASAC algorithm not only outperforms traditional methods like Q-learning and DQN but also demonstrates robust performance under varying network conditions, contributing to a more reliable and secure network optimization solution. The ability to adapt and maintain high security performance across different network scales and configurations provides strong support for future applications in network security and optimization.

However, the algorithm has certain limitations. One notable drawback is the lack of comparison between the ASAC algorithm and other currently optimized attack agent algorithms, which would provide a clearer benchmark for its relative performance. Despite its promising performance and stability, the algorithm may still face convergence speed issues in highly dynamic environments. Moreover, the parameter tuning process remains relatively complex, potentially increasing the difficulty of practical implementation. Future work should focus on developing more efficient adaptive parameter adjustment mechanisms to improve the ASAC algorithm's adaptability and robustness in even more complex and unpredictable environments.

References

[1] Greco C, Fortino G, Crispo B, et al. AI-enabled IoT penetration testing: state-of-the-art and research challenges[J]. Enterprise Information Systems, 2023, 17(9): 2130014.

[2] Ghanem M C, Chen T M, Nepomuceno E G. Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks[J]. Journal of Intelligent Information Systems, 2023, 60(2): 281-303.

[3] Wang Y, Li Y, Xiong X, et al. DQfD-AIPT: An Intelligent Penetration Testing Framework Incorporating Expert Demonstration Data[J]. Security and Communication Networks, 2023, 2023(1): 5834434.

[4] ZANG Y C, ZHOU T Y, ZHU J H, et al. Domain-Independent Intelligent Planning Technology and Its Application to Automated Penetration Testing Oriented Attack Path Discovery[J]. Journal of Electronics & Information Technology, 2020, 42(9): 2095-2107.

[5] Ma Q, Liu Y, Wu X S, Qu Y, Wang B L, Liu H R. Optimal Penetration Path Discovery Based on Value Iterative Algorithm[J]. Computer Systems and Applications, 2023, 32(12): 197-204.

[6] KANG Haiyan, LONG Molan, ZHANG Congming. Review on the Application of Automated Penetration Testing[J]. Journal of Cybersecurity, 2023, 1(2): 59-72.

[7] ZHOU T, ZANG Y, ZHU J, et al. NIG-AP: a new method for automated penetration testing[J]. Frontiers of Information Technology & Electronic Engineering, 2019, 20(9): 1277-1288.

[8] Schwartz J, Kurniawati H. Autonomous penetration testing using reinforcement learning[J]. arXiv preprint arXiv:1905.05965, 2019.

[9] Zhou S, Liu J, Zhong X, et al. Intelligent Penetration Testing Path Discovery Based on Deep Reinforcement Learning[J]. Computer Science, 2021, 48(07): 40-46.

[10] Nguyen H V, Teerakanok S, Inomata A, et al. The Proposal of Double Agent Architecture using Actor-critic Algorithm for Penetration Testing[C]//ICISSP. 2021: 440-449.

[11] Zeng Q, Zhang G, Xing C, et al. Intelligent Attack Path Discovery Based on Hierarchical Reinforcement Learning[J]. Computer Science, 2023, 50(07): 308-316.

[12] CVSS. https://www.first.org/cvss/v4.0/specification-document. Available online: Feb 1, 2024.