

Information Network

# ShotFlex: A reinforcement learning-based cyber attack path generation method for cybersecurity evaluation

Zhuocheng Yu<sup>1,2</sup>, Yan Jia<sup>1,\*</sup>, WeiHong Han<sup>1</sup>, Jiawei Zhang<sup>1</sup>, Mingsheng Yang<sup>1</sup>, and Yangyang Mei<sup>1,\*</sup> 

<sup>1</sup> Department of New Networks, Peng Cheng Laboratory, Shenzhen 518000, China

<sup>2</sup> Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China

Received: 28 February 2025 / Revised: 9 May 2025 / Accepted: 8 July 2025 / Published online: 28 July 2025

**Abstract** Penetration testing is an important method for discovering hidden vulnerabilities and attack paths in network systems, which is of great significance for evaluating network security. However, traditional penetration testing methods can only be carried out by security analysts, and the results are unstable, requiring extra time and money. Automated penetration testing can effectively reduce reliance on manual efforts. Automated attack planning, as one of the most critical components, has garnered widespread attention from researchers. Although previous studies have explored a variety of methods to mine attack paths, most of them require prior knowledge of the network topology, which contradicts reality and thus lacks application value. To automatically find the best potential attack path in complex and unknown networks from the hacker's perspective, this paper proposes ShotFlex: a reinforcement learning-based method that uses a quantifiable method to evaluate host and obtain rewards, which guides the agent to choose the best response action to discover attack paths from the intruder's perspective. ShotFlex also introduces a pruning strategy based on prior knowledge to accelerate path generation. Experimental results reveal that ShotFlex can combine current information to provide an effective decision and significantly improve the efficiency of penetration testing.

**Keywords** Cyber attack path generation, Cybersecurity evaluation, Reinforcement learning

**Citation** Yu Z, Jia Y, Han W, Zhang J, Yang M and Mei Y. ShotFlex: A reinforcement learning-based cyber attack path generation method for cybersecurity evaluation. Security and Safety 2025; 4: 2025006. <https://doi.org/10.1051/sands/2025006>

## 1 Introduction

With the continuous development of information technology, it is undeniable that the Internet has completely changed people's daily lives. Rapid development of network infrastructure has not only provided convenience, but also increased the threat of network vulnerabilities [1]. According to Skybox Security's "Vulnerability and Threat Trends Report 2024", the number of newly discovered vulnerabilities experienced an unprecedented growth trend in 2023. The timeline for vulnerability exploitation has been significantly shortened, while the average time to fix them remains too long. Faced with the frequent occurrence of cyber security incidents, ensuring the security of network information systems has become a widely recognized major challenge in both the academic and industrial communities [2].

\* Corresponding authors (email: [jiayan2020@hit.edu.cn](mailto:jiayan2020@hit.edu.cn) (Yan Jia); email: [1112006003@e.gzhu.edu.cn](mailto:1112006003@e.gzhu.edu.cn) (Yangyang Mei))

Traditional network defense strategies (*e.g.*, intrusion detection systems, access control, information encryption and vulnerability scanning) primarily focus on identifying and blocking external attacks to prevent potential threats from entering the internal network [3, 4]. Although these methods have been effective in past practical applications, this passive defense mechanism can only be addressed by relevant personnel after threats are detected, so it cannot prevent problems before occurring. Even the state-of-art intrusion detection system (IDS) still seems inadequate in the face of increasingly complex attacks. Therefore, it is crucial to develop proactive security strategies [5, 6].

Penetration testing (PT) is an active method widely used to assess the security of network systems. By simulating hacker attacks, PT tests potential security vulnerabilities in the target network without affecting the target system's network, thus achieving the goal of improving system security [7, 8].

Traditional penetration testing relies mainly on expert knowledge to infiltrate target networks. As network systems become increasingly complex and large, manual testing of this kind requires substantial time and labor costs, and the results are highly dependent on expertise and thus unstable [9, 10]. Automated penetration testing technology aims to enhance the level of automation in the penetration testing process. It can perform automated analysis of target systems to identify vulnerable nodes and potential attack paths within target network systems. Automated attack planning, as one of the most critical components, has attracted widespread attention from researchers [11].

In recent years, with the development of artificial intelligence (AI) technology, AI has been widely used and achieved in cyberspace security with its advantages of deep mining of potential pattern characteristics, strong generalization ability and relatively low cost (*e.g.*, ensemble learning-based malware detection [12], GNN-based intrusion detection system [13–15] and heterogeneous graph-based threat identification [16]). To automatically find the best potential attack path in complex and unknown networks from the hacker's perspective, this paper models the task based on reinforcement learning and proposes ShotFlex (**Short&Flexile** cyber attack path generation). The main contributions of this paper are as follows:

- (1) This paper develops a quantifiable method to evaluate the host state of the system, and uses it as a reward to guide the agent to choose the best response action at present.
- (2) This paper designs and implements a reinforcement learning-based cyber attack path generation method named ShotFlex. ShotFlex introduces a pruning strategy based on prior knowledge which can accelerate path generation.
- (3) This paper simplifies and builds three different scale network scenarios from the real world to verify the performance and effectiveness of ShotFlex. Experimental results show that ShotFlex can combine current information to provide an effective decision and significantly improve the efficiency of penetration testing.

The rest of paper is organized as follows: Existing reinforcement learning-based cyber attack path generation methods are shown in Section 2. Design ideas and details of ShotFlex are shown in Section 3. In Section 4, this paper designs and builds a simulation network scenario to discuss the ShotFlex's performance through experiments. In Section 5, the conclusions and future works are presented.

## 2 Related works

Reinforcement learning is a prominent branch of artificial intelligence technology, and has achieved significant success in fields such as Alpha Go [17] and Alpha Star [18] in gaming, autonomous driving [19] and robot control [20, 21]. Unlike supervised and unsupervised learning methods, which focus on learning latent features from data to minimize the loss function, reinforcement learning focuses on learning an agent policy which fundamentally aims to maximize the cumulative reward received by agent from environment, in order to learn the optimal policy for achieving specific target [22].

The basic workflow of reinforcement learning is illustrated in Figure 1. An agent observes the environmental state  $s_t \in S$  at time  $t$ , selects action  $a_t \in A$  according to the policy  $\pi(a_t|s_t)$ . The agent then takes the chosen action, transfers to the state  $s_{t+1}$ , and obtains a real-time reward  $r_t$ . The goal of agent is to maximize the cumulative reward  $R = \sum_{t=0}^{\infty} \gamma^t r_t$ , where  $\gamma \in (0, 1)$  is discount factor, representing the attention of agents to future rewards during learning. Throughout this iterative process, the agent continuously receives rewards, evolving and ultimately taking actions with the aim of maximizing rewards.

In penetration testing, pentesters make dynamic decisions according to the environmental state, the environmental state changed after the implementation of the action, and finally achieve the goal step by

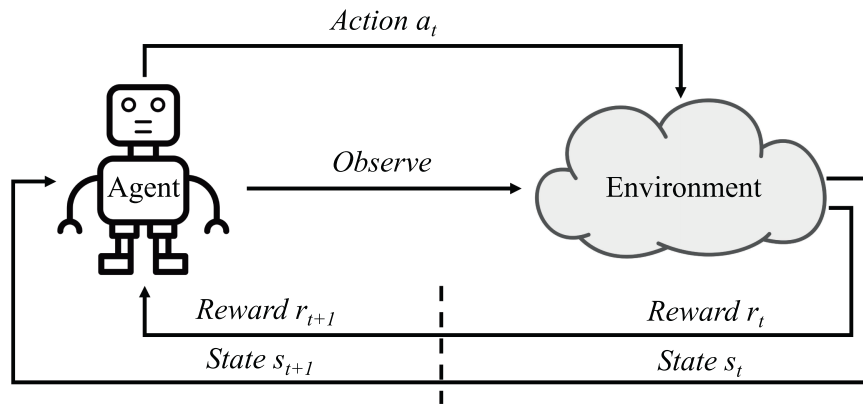


Figure 1. Basic reinforcement learning workflow

step, which is similar to the reinforcement of learning mechanism, so the researchers begin to consider applying reinforcement learning method to attack path generation task [23].

Greenwald *et al.* first applied reinforcement learning to automation penetration test in 2009 [24]. Zennaro *et al.* introduced Q-learning [25] algorithm to the capture-the-flag (CTF) and verified the feasibility of reinforcement learning in simple penetration testing tasks [26]. Yousefi *et al.* used MulVal [27] to generate an attack graph for a given network topology, and developed an algorithm named Q-learning-Attack-Graph which can find possible attack paths [28]. Erdödi L *et al.* focused on SQL injection, for the first time, SQL injection problem is modeled as a reinforcement learning task, and used Q-learning to solve SQL injection attack planning task [29]. Zhou *et al.* proposed a network information gain based on automated attack planning (NIG-AP) algorithm to discover attack paths automatically [30]. NIG-AP formalizes penetration testing as Markov decision process (MDP) [31], uses network information to obtain rewards, and guides agents to choose the best response action.

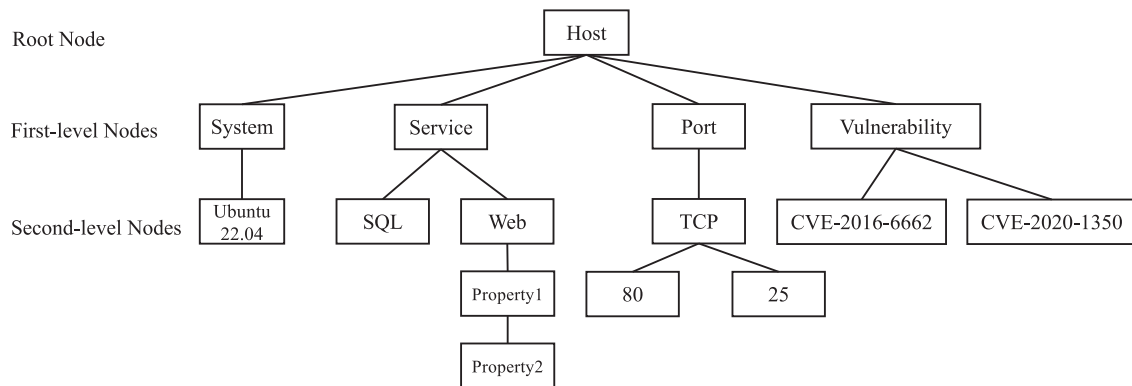
In penetration testing, the number of states after each action is unknown, the number of hosts connected to each host is uncertain. Therefore, the traditional table method will become impractical to store Q values. In addition, to ensure convergence of the optimal strategy, Q-learning needs to calculate each state-action pair at least once during learning. When the state space is very large, the application of Q-learning also faces plenty of challenges. In this case, researchers began to use function approximation (such as neural network) to represent Q-function, and Deep Q-Learning Network (DQN) is one of the most popular methods [32, 33]. Li *et al.* developed the INNES model based on MDP and DQN for automation penetration test [34]. Hu *et al.* combined attack tree and DQN to discover attack path which is the easiest to exploit [35]. However, this method is still limited by the dependency of the attack graph on global prior information, and cannot truly simulate the uncertainty of the penetration testing.

In addition, most advanced deep reinforcement learning methods (like DQN) cannot learn information of graphical structures and generalize, which means they can only run on the network topology seen during training but cannot run on a new topology. This indicates that these methods can only be used in a static known network system and limit potential applications in reality.

Rather than modeling penetration testing as MDP, studies [36–39] consider penetration testing as partially observable Markov decision process (POMDP) [40]. POMDP allows only partial observation of the system state at each time, but not complete acquisition of the current state. POMDP can model the uncertainty well in the penetration test, while it brings about an increase in computational complexity, which is difficult to apply for the large-scale network scenarios. Therefore, how to efficiently guide the agent to learn action policies in a sparse and unknown state space without obtaining global network topology is the key point in applying reinforcement learning to attack path generation task.

### 3 Proposed method

This section proposes a reinforcement learning-based method to generate cyber attack paths. To better simulate the real scenarios in penetration testing where the global network topology is unknown, this



**Figure 2.** Overview of ShotFlex

paper proposes the ShotFlex (Short&Flexile cyber attack path generation) based on Monte Carlo Tree Search (MCTS) [41]. This paper integrates expert knowledge to guide agents to efficiently learn action strategies in a sparse and unknown state space. The overall framework of ShotFlex is shown in Figure 2.

### 3.1 Model as MDP

ShotFlex models the cyber attack path generation task as a MDP which can be defined by a tuple  $\langle S, A, R, P \rangle$ , where  $S$  represents the state set,  $A$  represents the action set,  $P(s'|s, a)$  represents the probability of transition to  $s'$  after taking the action  $a$  at state  $s$ ;  $R(s, a)$  represents the reward obtained by taking the action  $a$  in state  $s$ . Reinforcement learning introduces a reward function to calculate the reward. The target of ShotFlex is to maximize the total rewards in limited steps, a higher reward means a better path ShotFlex get, which will be discussed later.

**State.** The state space  $S = \{s_1, s_2, \dots\}$  is composed of hosts in the network, each  $s$  denotes a single host. Each host plays a different role in network and provides different services. The host currently accessed by pentesters in the network is the state  $s_t$  of an agent in time  $t$ , and the terminal state is the target host in penetration test. The state of environment is usually represented by vectors, ShotFlex uses a word2vec-based method to encode each host  $s$  [42].

Specifically, a tree structure named a host-tree is utilized to denote a host as shown in Figure 3. The first step to construct host-tree is choosing the root node, which represents the observed host. The second step is appending child nodes (called First-level Nodes) to the root node. The first-level nodes represent the categories of host properties, and in ShotFlex the operating system, the running services' names, the opened ports and vulnerabilities are chosen to describe a host. Next step is appending further child nodes (called Second-level Nodes) that represent detailed information about each first-level nodes' properties, meanwhile the second-level nodes can also have more successor nodes. It is worth noting that one host may usually provide multiple services, for example in Figure 3, there are SQL server and Web server running in the host, so a node can also have multiple child nodes.

The port number here is not a continuous numerical value, so there are only discrete words in host-tree. ShotFlex uses word2vec to generate the basic embedding vectors for each node, and after constructing host-trees ShotFlex uses algorithm 1 to get each host-tree's vector representation. Since different hosts in network usually have different nodes and large number of nodes, directly concatenating all basic vectors of each node will lead to inconsistent dimensions and will cost huge computing resources. To address this challenge, ShotFlex introduces t-SNE [43] which transforms the similarity between data points into joint probability, and minimizes the kullback-leibler divergence difference between low-dimensional embedding and high-dimensional data joint probability. In this way, ShotFlex can unify the dimension of each host vector, identify and match each host, and reduce computational cost.

**Action.** In reinforcement learning, the action space  $A$  is the set of all actions that an agent can take. After observing the current environment state, the agent takes actions to the environment according to the policy and gets rewards. The agent's selection of actions is decided by current environment state and policy, and the execution of actions will further change the current environment state.

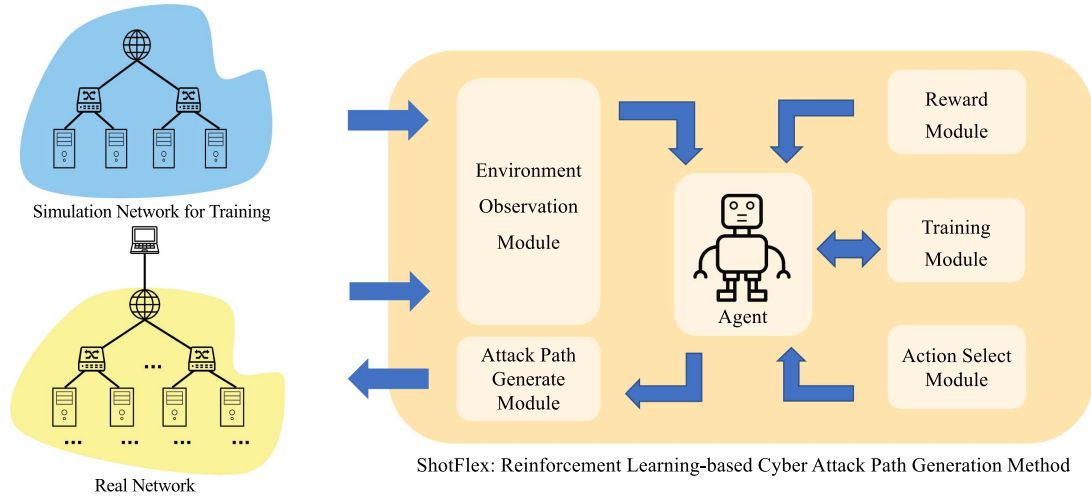


Figure 3. Example of a host-tree

---

**Algorithm 1** Host-trees vectorization
 

---

```

1: Input: host-trees list: $hts$ , dimension of host vector: $n$ , word2vec lookup table: $emb$  (node value: basic embedding vector)
2: Output: vectors table of host: $V$  (host: host vector)
3:  $V = []$ ; // init as an empty hash table
4: for  $ht$  in  $hts$  do
5:    $temp\_v = []$ ; // init as an empty host vector
6:    $nodes = DFS(ht)$ ; // get the DFS sequence list of host-tree
7:    $current\_host = nodes[0]$ ; // the root of host-tree is current host
8:   for  $node$  in  $nodes[1:]$  do
9:      $temp\_v = temp\_v.concatenate(emb[node])$ ; // concatenate each node's embedding
10:  end for
11:   $temp\_v = t-SNE(temp\_v, n)$ ; // unify dimension
12:   $V[current\_host] = temp\_v$ ; // append to list
13: end for
14: return  $V$ ; // return
    
```

---

Considering a network with  $M$  hosts and  $N$  actions can be chosen, each time the agent makes a decision, the size of solution space to search is  $O(M * N)$ . The size of action space will directly affect the learning efficiency of agent, meanwhile a large action space also reduces the convergence efficiency of algorithm. ShotFlex considers agent's actions which jumps between current host and next host, that is, the vulnerability exploitation in hosts. One vulnerability exploitation corresponds to only one action. One host may have multiple vulnerabilities, so that one host may correspond to multiple actions.

ShotFlex does not introduce scanning as a single action, because ShotFlex focuses on the cyber attack path generation task and scanning is indispensable in real penetration testing. Based on the assumption that the host information of next state has been investigated, the only task of ShotFlex is to choose which vulnerability should be chose to exploit. In this way, ShotFlex can greatly simplify the problem size and speed up learning the optimal policy.

**Reward.** The reward function  $R(s_t, a_t)$  is used to quantify the immediate reward  $r_t$  obtained by agent after executing action  $a_t$  at state  $s_t$ . Reinforcement learning uses the cumulative reward value to evaluate the advantages and disadvantages of a policy  $\pi$ . The target of training is to find an optimal policy  $\pi^*$  to generate an attack path that can maximize the total reward. It can be formulated as Equation (1).

$$\pi^* = \arg \max_{\pi} [sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)] \quad (1)$$

Different reward values are given for different vulnerability exploitation in ShotFlex so that agent can learn the optimal vulnerability exploitation through iterative training. ShotFlex introduces a CVSS based vulnerability quantitative evaluation method to define the rewards for each vulnerability, and

**Table 1.** Vulnerability quantification score table

CVE ID	Vulnerability exploitability score
CVE-1999-0001	10
CVE-1999-0002	10
...	...
CVE-2025-3847	7.3

**Table 2.** Success rate of different vulnerability levels

Vulnerability reliability level	Success rate(%)
Excellent	90
Great	80
Good	70
Normal	60
Average	50
Low	30
Manual	10

**Table 3.** Host asset level quantification score table

Host type	Host risk level	Host score
DNS server, Backbone router, Database server	HIGH	9
Mail server, FTP server, Host for storing key information, etc.	MEDIUM-HIGH	7
Web server, Firewall, Key location host	MEDIUM	5
Normal host, etc.	MEDIUM-LOW	3
Temp host, etc.	LOW	1

guides agent to learn the optimal attack path by introducing expert prior knowledge. This quantitative evaluation method combines vulnerability availability score, host asset score and historical experience.

The vulnerability quantification score is shown in Table 1, the vulnerability exploitability score is in the range from 0 to 10, which is the basement of reward function. In order to better simulate the uncertainty of attacks in the real world, ShotFlex introduces the probability of successful vulnerability exploitation which is shown in Table 2 [44]. If a vulnerability is successfully exploited, the agent will get an immediate reward. Otherwise, the agent will get a punishment which is equal to the negative vulnerability exploitability score. Further considering that different hosts often have different values, ShotFlex introduces host asset level quantification score to address this problem which is shown in Table 3. Meanwhile, to avoid the agent moving only in one subnet, ShotFlex encourages agent to execute lateral movement and set a separate reward *Vulnerability Exploitability Score* for this purpose. If the agent moves to a subnet that has never been accessed before, the  $\alpha$  is 1 otherwise is 0.

Based on the above considerations, ShotFlex defines the immediate reward as Equation (2).

$$R(s_t, a_t) = \begin{cases} (Host\ Score * Vulnerability\ Exploitability\ Score) + \\ \quad \alpha * Lateral\ Movement\ Score & \text{successfully exploitation} \\ -Host\ Score * Vulnerability\ Exploitability\ Score & \text{otherwise} \end{cases} \quad (2)$$

### 3.2 Monte Carlo Tree Search in ShotFlex

Monte Carlo Tree Search (MCTS) is a simulation-based algorithm, which is used to search for the optimal action in large-scale state space. MCTS can solve the problem of unknown probability distribution of each step, which is consistent with the execution of each action in cyber attack path generation task. To solve

MDP, MCTS algorithm estimates the optimal policy by sampling possible future decision paths. The basic idea of MCTS is to make full use of known information while ensuring certain exploration. The MCTS includes four steps: **Selection**, **Expansion**, **Simulation** and **Backpropagation**.

**Selection.** Start from the root node (the initial state  $s_0$ ), select child nodes according to the policy in the tree until a leaf node (target state means that the target host is accessed) is reached. MCTS shows an asymmetric tree growth to adapt the topology of the search space. MCTS will visit the more 'interested' nodes and focus the search space on more relevant parts. This characteristic of MCTS means that it eventually will find those more excellent actions and focus on the search there. Therefore, according to different reward settings, MCTS will focus on searching the path that can maximize the reward. Action selections are based on the Upper Confidence Bound for Trees (UCB), which can be formulated as Equation (3).

$$a = \arg \max_{a \in A} [Q(s, a) + c * \sqrt{\frac{\ln N(s)}{N(s, a)}}] + w * P(s, a) - \alpha * depth \quad (3)$$

Where  $Q(s, a)$  is the value of action  $a$  in current state  $s$ ,  $N(s)$  is the number of times state  $s$  is accessed,  $N(s, a)$  is the number of times to select action  $a$  in state  $s$ ,  $c$  is the hyper parameter,  $P(s, a)$  is an extensional item based on prior knowledge to describe the priority of selecting action  $a$  in state  $s$  and  $depth$  reflects the length of attack length.

**Expansion.** If the selected leaf node is not the target state, expand a new child node  $c$  according to the current policy. In expansion, ShotFlex prioritizes expanding the most effective actions based on vulnerability quantification score and prior knowledge, which can help to reduce ineffective action extensions and make the search tree more efficient in focusing on potential optimal attack paths.

**Simulation.** Start from the child node  $c$  and execute a rollout, until access to the target state or the path length reaches maximum. The accumulated reward  $G$  can be formulated as Equation (4), where  $t$  is the time step of this simulation.

$$G = \begin{cases} \sum_{t=0}^T \gamma^t R(s_t, a_t) & \text{access to the target} \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

While due to the decision space is usually large and unknown in real penetration testing, key nodes may not be visited enough in the exploration, which may lead MCTS to the failure to make good decisions within a certain period of time.

To address this challenge, ShotFlex introduces real historical attack data and heuristic simulation policies to replace random simulations. In this way, the result of rollout will be more similar to the pentesters' actions, which means that the results will become more realistic than random simulations, and nodes will produce truly reliable estimates after fewer iterations.

**Backpropagation.** Backpropagation traces the simulated cumulative reward  $G$  back to all nodes along the path and updates each  $Q(s, a)$  and  $N(s, a)$  as Equation (5).

$$\begin{aligned} Q(s, a) &\leftarrow \frac{\sum G}{N(s, a)} \\ N(s, a) &\leftarrow N(s, a) + 1 \end{aligned} \quad (5)$$

## 4 Results and experimental analysis

### 4.1 Experimental scenario

This paper builds a network topology to test the effectiveness of ShotFlex, which is simplified from a real medium-scale network system. The overall network topology is shown in Figure 4.

The 192.168.8.0/24 network segment is DMZ area which is used to provide external services and can be accessed by external Internet. Two servers (192.168.8.2 and 192.168.8.5) are deployed in DMZ area to provide mail and website services, respectively, allowing Internet users to access ports 25 and 80 through TCP protocol.

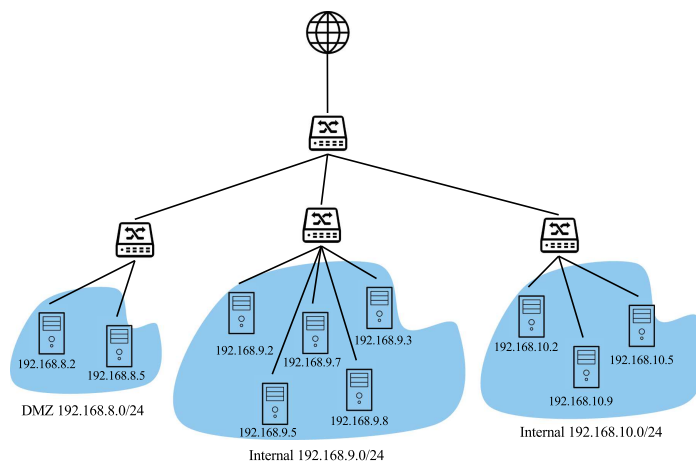


Figure 4. Simulated network topology with 10 hosts

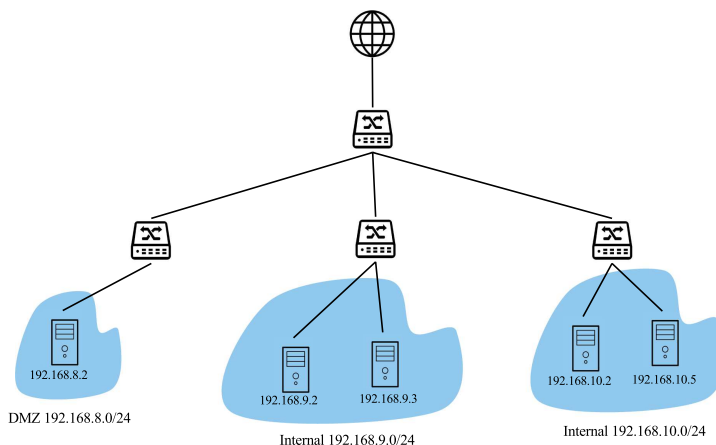


Figure 5. Simulated network topology with 5 hosts

The 192.168.9.0/24 and 192.168.10.0/24 network segments are internal network areas, these two areas cannot be accessed by external Internet directly, while the same subnet can access all ports mutually. In 192.168.9.0/24 network segment, the 192.168.9.3 server provides MySQL service and 192.168.9.8 server provides HTTP service. These servers open 3306 port and 80 port to other hosts in DMZ area and internal networks, meanwhile allowing cross-subnet access. The hosts in 192.168.10.0/24 network segment store the asset data, so the target host in this scenario is set **192.168.10.2**.

To test the performance of ShotFlex at different network scales, this paper expands two additional scenarios based on the above topology which are shown as Figure 5 and Figure 6. Corresponding to small network systems and large network systems, respectively, and the target host are **192.168.10.5** and **192.168.15.2**, respectively. The vulnerability information of each host is shown in Table 4.

## 4.2 Experimental results and analysis

The entire code execution environment is based on python3.8.8 run on the server with 3 RTX4090. This paper sets the UCB’s parameter  $c$  to 1, which encourages the search more inclined to the path with high reward, rather than overemphasizing the exploration of new paths. Meanwhile, the parameters  $w$  and  $\alpha$  are set to 1. Considering the different scenario’s network topology scales, the maximum path length  $T$  is 3, 4 and 7, respectively for 5-Hosts, 10-Hosts and 20-Hosts. The simulation number is 1000, and ShotFlex repeats 10 times for each scenario. The general results are shown in Table 5.

In 10-Host scenario, ShotFlex generates a attack path **External Internet**  $\rightarrow$  **192.168.8.5**  $\rightarrow$  **192.168.9.3**  $\rightarrow$  **192.168.10.5**  $\rightarrow$  **192.168.10.2**. Pentesters first determine whether the httpd version is

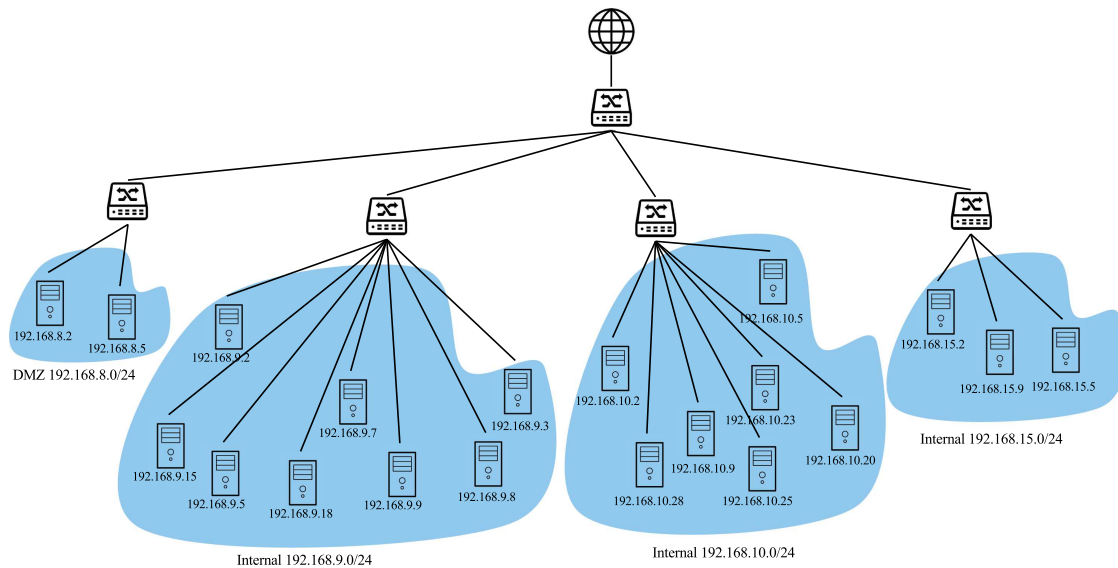


Figure 6. Simulated network topology with 20 hosts

Table 4. Vulnerability information

Host	Vulnerability ID	Vulnerability exploitability score	Vulnerability level	Host score
192.168.8.2	CVE-2013-2566	5.9	Excellent	5
192.168.8.5	CVE-2023-25690	9.8	Excellent	5
192.168.9.2	CVE-2016-2183	7.5	Good	3
192.168.9.3	CVE-2016-6662	9.8	Normal	9
192.168.9.5	CVE-2020-1350	10.0	Excellent	3
192.168.9.7	CVE-2013-2566	5.9	Excellent	5
192.168.9.8	CVE-2023-25690	9.8	Manual	5
192.168.9.9	CVE-2020-1350	10.0	Excellent	3
192.168.9.15	CVE-2019-10081	7.5	Normal	5
192.168.9.15	CVE-2019-10082	9.1	Normal	5
192.168.9.18	CVE-2020-1934	5.3	Normal	5
192.168.10.2	CVE-2016-2183	7.5	Good	3
192.168.10.5	CVE-2020-1350	10.0	Excellent	5
192.168.10.9	CVE-2015-2808	5.0	Good	5
192.168.10.20	CVE-2021-3711	9.8	Good	9
192.168.10.20	CVE-2021-3712	7.4	Good	9
192.168.10.23	CVE-2019-0708	9.8	Excellent	3
192.168.10.23	CVE-2020-1350	10.0	Excellent	3
192.168.10.25	CVE-2020-13950	7.5	Good	5
192.168.10.28	CVE-2021-36160	7.8	Good	5
192.168.10.28	CVE-2021-33193	7.5	Good	5
192.168.15.2	CVE-2016-2183	7.5	Good	3
192.168.15.5	CVE-2020-1350	10.0	Excellent	3
192.168.15.9	CVE-2015-2808	5.0	Great	7

less than 2.4.55 or not, and then execute Internal HTTP Request Smuggling via Header Injection. Through CVE-2023-25690, pentesters can access the host 192.168.8.5. Then, pentesters execute the Lateral Movement from 192.168.8.0/24 to 192.168.9.0/24 through CVE-2016-6662 and get to the host 192.168.9.3. Then, pentesters can send specially crafted request packets to Windows DNS Server 192.168.10.2, aiming to exploit vulnerability CVE-2020-1350 and complete lateral movement to 192.168.10.0/24. Once pentesters access to the 192.168.10.0/24, they can get the host 192.168.10.2 through CVE-2016-2183.

**Table 5.** Results of different scenarios

Scenario	Generated path	Total reward
10-Hosts	External Internet $\rightarrow$ 192.168.8.5 $\rightarrow$ 192.168.9.3 $\rightarrow$ 192.168.10.5 $\rightarrow$ <b>192.168.10.2</b>	240.7303
5-Hosts	External Internet $\rightarrow$ 192.168.8.2 $\rightarrow$ 192.168.9.3 $\rightarrow$ <b>192.168.10.5</b>	208.6996
20-Hosts	External Internet $\rightarrow$ 192.168.8.5 $\rightarrow$ 192.168.9.3 $\rightarrow$ 192.168.9.8 $\rightarrow$ 192.168.10.5 $\rightarrow$ 192.168.10.20 $\rightarrow$ 192.168.15.9 $\rightarrow$ <b>192.168.15.2</b>	376.5458

**Table 6.** Time performances of different scenarios (ms)

Scenario	Time	Time without knowledge	Time of DFS	Steps of PentestGPT
10-Hosts	<b>17.1197</b> ( $\pm$ <b>1.9626</b> )	21.5669( $\pm$ 0.8836)	20.4378( $\pm$ 3.5844)	12
5-Hosts	16.1153( $\pm$ 0.7392)	18.2423( $\pm$ 0.4017)	<b>1.8087</b> ( $\pm$ <b>0.3555</b> )	7
20-Hosts	<b>19.4549</b> ( $\pm$ <b>1.6413</b> )	24.2723( $\pm$ 0.4469)	77484.7892( $\pm$ 4992.6695)	19

By analyzing the attack path of the 10-Host scenario, we can discover that ShotFlex has certain ability to generate the most threatening attack path within limited steps. In fact, the attack path generated by ShotFlex is almost consistent with the choice pentesters in the real world. The reason is that the target of pentesters is to get as close to the target host as possible and obtain more assets. Meanwhile, due to the limitation of must to reach the target node, ShotFlex will not blindly attempt to exploit high-value vulnerabilities to access the host. That is why in the 5-Host scenario ShotFlex selects the host 192.168.10.5 rather than the higher value host 192.168.10.2.

In practice, we find that ShotFlex’s strategy is more aggressive, as it prefers to attempt to penetrate higher value hosts, even if this may introduce a more greater risk. And we can easily find ShotFlex tries its best to make use of limited steps in each scenario, because this will get a higher reward. This strategy is acceptable in penetration testing, while ShotFlex also provides a flexible method to control the attack path generation strategy by changing  $\alpha$  in Equation (3). That is, a higher  $\alpha$  will result in a lower reward for the current action. As the path length increases, the immediate reward can even become negative.

Table 6 shows the time performance of ShotFlex in different scenarios. The results show that ShotFlex can make decisions based on a series of prior knowledge and generate an attack path with a stable time performance. In 10-Host scenario, ShotFlex can generate a 4-hop path in 17.1197 ms on average, compare to using MCTS directly (without priori knowledge) needs 21.5669 ms on average. In three different network scale scenarios, the pruning strategy reduces running time with 20.6204%, 11.6597% and 19.8473%, respectively. And as the network scale increases, this acceleration effect becomes more evident. However, it also brings the disadvantage that the time performance becomes more unstable, which may be because the strategy is not always effective.

Performing a full traversal of the network topology and solving for the reward of each path can also find the optimal path. We use DFS to traverse the graph and obtain all paths. The result shows that in small network scale (like 5-Hosts) DFS has significant performance advantages, it only takes 1.8087 ms on average, which is far lower than ShotFlex. This is because MCTS has a series of initialization and computation processes, which are not cost-effective in small-scale networks. As the size of the network increases, the cost of traversing all hosts to generate paths becomes unacceptable, using DFS method to solve in 20-Hosts scenario requires 77484.7892 ms on average.

Large Language Models (LLMs) have developed rapidly in recent years and have achieved great success in many fields. PentestGPT [45] is an open source end-to-end automated penetration testing framework powered by LLMs. We also test PentestGPT in three scenarios, although PentestGPT can successfully guide pentesters to access the target host, we still find it has several shortcomings in cyber-attack path generation task. (1) Due to the different target tasks, PentestGPT needs more operation steps to get the attack path, although we have informed PentestGPT that all network topology and configuration information only require it to generate the final attack path. PentestGPT requires step-by-step execution

of attack tools (such as scanning ports, brute-force, etc.). As in 10-Hosts scenario we totally conduct 12 rounds of dialogue to get the final attack path, and 7 rounds for 5-Hosts scenario, 19 rounds for 20-Hosts scenario. (2) PentestGPT can generate cyber-attack path, the results are difficult to reproduce stably and need to extra rank to evaluate paths. (3) The effectiveness of the LLM-based method is directly related to the back-end models, and further research is needed to develop a LLM-based attack path generation framework due to the inherent deficiencies of LLMs. Nevertheless, we remain confident in LLMs' potential and are committed to conducting further research in attack path generation task.

## 5 Conclusions and future works

This paper proposes a reinforcement learning-based cyber attack path generation method: ShotFlex. ShotFlex uses a quantifiable method to evaluate the host and models the cyber attack path generation task as a Markov Decision Process. ShotFlex uses Monte Carlo Tree Search to search the best attack path and introduces a pruning strategy to accelerate path generation. To verify the performance and effectiveness of ShotFlex, this paper simplifies and builds three different scale network scenarios from the real world. Through the experiment, this paper verifies ShotFlex's basic function and performance, and preliminarily explores ShotFlex's potential in the cyber attack path generation task in the real world.

In future research work, we will focus mainly on the following research directions. (1) In recent years, large language models (LLMs) have been widely applied in various fields and have achieved great success, leaving a strong impression on us in practice. We believe that with the incredible summarising ability and reasoning ability of LLMs, LLMs must have great potential for application in the cyber attack paths generation task [45, 46]. (2) In addition to introducing the LLM technology, we also consider using hierarchical reinforcement learning to train multiple agents, and combine ATT&CK to guide agents to learn more realistic attack paths [47–49]. (3) For the cyber attack paths generation task, we will focus on developing more advanced reinforcement learning algorithms to further reduce computational costs and achieve better performance.

### Acknowledgments

We thank Dr. Dong for his help in polishing the paper, the reviewers for their thoughtful suggestions, and the editors' dedicated work.

### Funding

This work was supported by the Major Key Project of PCL (Grant No. PCL2024A05-3).

### Conflicts of interest

The authors declare there is no conflict of interest.

### Data availability statement

No data are associated with this article.

### Author contribution statement

Z.Y. and Y.J. conceptualized, wrote the paper and oversaw all aspects of the revisions. Y.M. and M.Y. summarized the related works and analyzed the results. W.H. and J.Z. conducted the experiments. All authors reviewed the manuscript.

## References

- [1] Chen Z. Research on internet security situation awareness prediction technology based on improved RBF neural network algorithm. *J Comput Cognit Eng* 2022; **1**: 103–08.
- [2] Verma R, Kumari A, Anand A et al. Revisiting shift cipher technique for amplified data security. *J Comput Cognit Engineering* 2024; **3**: 8–14.
- [3] Zheng Y, Li Z, Xu X et al. Dynamic defenses in cyber security: Techniques, methods and challenges. *Digital Commun Networks* 2022; **8**: 422–35.
- [4] Chen Z, Kang F, Xiong X et al. A survey on penetration path planning in automated penetration testing. *Appl Sci* 2024; **14**: 8355.
- [5] Han X, Pasquier T, Seltzer M. Provenance-based intrusion detection: opportunities and challenges. In: 10th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2018), 2018.

- [6] Li Z, Chen QA, Yang R et al. Threat detection and investigation with system-level provenance graphs: A survey. *Comput Secur* 2021; **106**: 102282.
- [7] Stefinko Y, Piskozub A, Banakh R. Manual and automated penetration testing. Benefits and drawbacks. Modern tendency. In: 2016 13th international conference on modern problems of radio engineering, telecommunications and computer science (TCSET). IEEE 2016, 488–91.
- [8] Zhou S, Liu J, Zhou X et al. Intelligent Penetration Testing Path Discovery Based on Deep Reinforcement Learning. *Comput Sci* 2021; **48**: 40–6 (in Chinese).
- [9] Bertoglio DD, Gil A, Acosta J et al. Towards new challenges of modern Pentest. In: International conference on WorldS4, Singapore, Springer Nature Singapore, 2023, 21–33.
- [10] Chen K, Lu H, Fang BX et al. Survey on automated penetration testing technology research. *Ruan Jian Xue Bao/J Software* 2024; **35**: 2268–88 (in Chinese).
- [11] Polatidis N, Pavlidis M, Mouratidis H. Cyber-attack path discovery in a dynamic supply chain maritime risk management system. *Comput Standards Interfaces* 2018; **56**: 74–82.
- [12] Yu Z, Li S, Bai Y et al. REMSF: a robust ensemble model of malware detection based on semantic feature fusion. *IEEE Int Things J* 2023; **10**: 16134–143.
- [13] Rehman MU, Ahmadi H, Hassan WU. FLASH: A comprehensive approach to intrusion detection via provenance graph representation learning. In: 2024 IEEE Symposium on Security and Privacy (SP). IEEE Computer Society, 2024, 139–39.
- [14] Chen T, Dong C, Lv M et al. APT-KGL: An intelligent APT detection system based on threat knowledge and heterogeneous provenance graph learning. *IEEE Trans Dependable Secure Comput*, 2022, 1–15.
- [15] Wang S, Wang Z, Zhou T et al. Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning. *IEEE Trans Inform Forensics Secur* 2022; **17**: 3972–987.
- [16] Gao Y, Li X, Peng H et al. Hincti: A cyber threat intelligence modeling and identification system based on heterogeneous information network. *IEEE Trans Knowledge Data Eng* 2020; **34**: 708–22.
- [17] Silver D, Huang A, Maddison CJ et al. Mastering the game of Go with deep neural networks and tree search. *Nature* 2016; **529**: 484–89.
- [18] Vinyals O, Babuschkin I, Czarnecki WM et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 2019; **575**: 350–54.
- [19] Zhang HJ, Zhao J, Wang R et al. Multi-objective reinforcement learning algorithm and its application in drive system. In: Proc. 34th Annu. IEEE Conf. Ind. Electron., Orlando, FL, USA, 2008, 274–79.
- [20] Tang C, Abbatematteo B, Hu J, et al. Deep reinforcement learning for robotics: A survey of real-world successes[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2025; **39**: 28694–28698.
- [21] Arulkumaran K, Deisenroth MP, Brundage M et al. Deep reinforcement learning: A brief survey. *IEEE Signal Process Mag* 2017; **34**: 26–38.
- [22] Sutton RS, Barto AG. Reinforcement Learning: An introduction, MIT press, 2018.
- [23] Greenwald L, Shanley R. Automated planning for remote penetration testing. In: Proc. of the 2009 IEEE Military Communications Conf., Boston, IEEE, 2009, 1–7.
- [24] Alhamed M, Rahman MMH. A systematic literature review on penetration testing in networks: future research directions. *Appl Sci* 2023; **13**: 6986.
- [25] Watkins CJCH, Dayan P. Q-learning. *Mach Learn* 1992; **8**: 279–92.
- [26] Massimo Zennaro F, Erdodi L. Modeling penetration testing with reinforcement learning using capture-the-flag challenges: trade-offs between model-free learning and a priori knowledge. arXiv preprint arXiv:2005.12632, 2020.
- [27] Ou X, Govindavajhala S, Appel AW. MulVAL: A logic-based network security analyzer. In: Proceedings of the 14th Conference on USENIX Security Symposium Volume 14, Baltimore, MD, USA, 31 July–5 August 2005, 8.
- [28] Yousefi M, Mtetwa N, Zhang Y et al. A reinforcement learning approach for attack graph analysis. In: 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), IEEE, 2018, 212–17.
- [29] Erdodi L, Sommervoll, Zennaro FM. Simulating SQL injection vulnerability exploitation using Q-learning reinforcement learning agents. *J Inform Secur Appl* 2021; **61**: 102903.
- [30] Zhou T, Zang Y, Zhu J et al. NIG-AP: A new method for automated penetration testing. *Front Inform Technol Electron Eng* 2019; **20**: 1277–88.
- [31] Puterman ML. Markov decision processes: discrete stochastic dynamic programming, John Wiley & Sons, 2014.
- [32] Mnih V, Kavukcuoglu K, Silver D et al. Human-level control through deep reinforcement learning. *Nature*, 2015; **518**: 529–33.
- [33] Mnih V. Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602, 2013.

- [34] Li Q, Hu M, Hao H et al. INNES: An intelligent network penetration testing model based on deep reinforcement learning. *Appl Intell* 2023; **53**: 27110–127.
- [35] Hu Z, Beuran R, Tan Y. Automated penetration testing using deep reinforcement learning. In: 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), IEEE, 2020, 2–10.
- [36] Sarraute C, Buffet O, Hoffmann J. POMDPs make better hackers: Accounting for uncertainty in penetration testing. *Proc AAAI Conf Artif Intell* 2012; **26**: 1816–24.
- [37] Shmaryahu D, Shani G, Hoffmann J et al. Simulated penetration testing as contingent planning, *Proc Int Conf Automated Planning Sched* 2018; **28**: 241–49.
- [38] Ghanem MC, Chen TM. Reinforcement learning for intelligent penetration testing. In: 2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), IEEE, 2018, 185–92.
- [39] Schwartz J, Kurniawati H, El-Mahassni E. Pomdp+ information-decay: Incorporating defender's behaviour in autonomous penetration testing. *Proc Int Conf Automated Planning Sched* 2020; **30**: 235–43.
- [40] Kaelbling LP, Littman ML, Cassandra AR. Planning and acting in partially observable stochastic domains. *Artif Intell* 1998; **101**: 99–134.
- [41] Coulom R. Efficient selectivity and backup operators in Monte-Carlo tree search. In: *International Conference on Computers and Games*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2006, 72–83.
- [42] Mikolov T. Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781, 2013, 3781.
- [43] Maaten L, Hinton G. Visualizing data using t-SNE[J]. *Journal of machine learning research*, 2008; **9**: 2579–2605.
- [44] Hu T, Zang Y, Cao R et al. Research on attack path discovery algorithm based on multi-heuristic information fusion. *J Cyber Secur* 2021; **6**: 202–11 (in Chinese).
- [45] Deng G, Liu Y, Mayoral-Vilches V et al. PentestGPT: Evaluating and harnessing large language models for automated penetration testing. In: 33rd USENIX Security Symposium (USENIX Security 24), 2024, 847–64.
- [46] Shen X, Wang L, Li Z et al. PentestAgent: Incorporating LLM Agents to Automated Penetration Testing, arXiv preprint arXiv:2411.05185, 2024.
- [47] Hasegawa K, Hidano S, Fukushima K. AutoRed: Automating red team assessment via strategic thinking using reinforcement learning. In: *Proceedings of the Fourteenth ACM Conference on Data and Application Security and Privacy*, 2024, 325–36.
- [48] Singh AV, Rathbun E, Graham E et al. Hierarchical multi-agent reinforcement learning for cyber network defense, arXiv preprint arXiv:2410.17351, 2024.
- [49] Schmid M, Moravcik M, Burch N et al. Student of games: A unified learning algorithm for both perfect and imperfect information games. *Sci Adv* 2023; **9**: eadg3256.



**Zhuocheng Yu** received the M.S. degree in Guangzhou University, China, in 2024. He is currently pursuing the Ph.D. degree in Southern University of Science and Technology and Peng Cheng Laboratory, China. His research interests include cyber attack path generation and graph machine learning with applications in cyberspace security.



**Yan Jia** is a professor with the Department of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), China. She serves as an executive Director with Chinese Information Processing Society of China and the chair of Specialty Committee of Big Search in Cyberspace. Her research interests cover big data analysis, artificial intelligence, online social network analysis and security situation awareness and analysis in cyberspace.



**Weihong Han** received her M.S. degree in National University of Defense Technology, China, in 1997 and her Ph.D. degree in National University of Defense Technology in 2000. Now, she is a researcher in the Department of New Networks, Peng Cheng Laboratory, China. Her current research includes computer network and network security.



**Jiawei Zhang** received his Bachelor's degree from Xidian University, China, in 2017 and subsequently earned his Ph. D. degree in Computer Science from Fudan University in 2022. He currently holds the position of assistant professor in the Department of New Networks at Peng Cheng Laboratory. His research expertise lies in the field of deep learning and its practical applications in intrusion detection systems.



**Mingsheng Yang** received his M.S. degree in University of Electronic Science and Technology of China in 2007. Now, he is an engineer in the Department of New Networks, Peng Cheng Laboratory, China. His current research focuses on network security.



**Yangyang Mei** received his M.S. degree in Guangdong Polytechnic Normal University, China, in 2019 and his Ph.D. degree in Guangzhou University in 2024. Now, he is a postdoctor in the Department of New Networks, Peng Cheng Laboratory, China. His current research includes machine learning, cyberspace security and cyber security situational awareness.