Master's Research Project Report

Assessment of Reinforcement Learning-Based Penetration Testing Methodologies

CHEN Zhi

Supervisor BEURAN Razvan Florin

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Information Science)

March, 2025

Abstract

With the increasing complexity and frequency of cyber attacks, traditional penetration testing methods relying on manual operations are becoming increasingly inadequate. Penetration testing is a technique that simulates the behavior of an attacker to identify security vulnerabilities in a system. Traditional penetration testing often relies on manual operations, which are time-consuming and expensive, and the test results are highly dependent on the experience and skills of the testers.

To improve the efficiency and effectiveness of penetration testing, researchers are exploring the application of reinforcement learning in the field of penetration testing. Reinforcement Learning is a branch of machine learning, that involves an agent interacting with the environment and continuously learning the optimal strategy to maximize cumulative rewards. In penetration testing, reinforcement learning can automate the process of discovering and exploiting system vulnerabilities, thereby improving the coverage and depth of the tests. In recent years, many research efforts have demonstrated the potential of reinforcement learning in automated penetration testing.

This research report provides a comprehensive review and analysis of the application of reinforcement learning in automated penetration testing. We first conduct an in-depth analysis of around 30 relevant papers to identify the main problems and challenges in current research, as well as summarize the root causes and impacts of these issues. This process provides a reference for future research, helping to avoid repetitive mistakes and improve the effectiveness and precision of studies.

In addition, designing an evaluation method and criteria based on experimental results is essential for assessing the effectiveness of automated penetration testing. To this end, we develop a reliable and feasible evaluation method and standard based on our literature analysis. Furthermore, we conduct a series of experiments using the Network Attack Simulator (NAS) to ensure the reliability and credibility of the results.

The conclusions of this research demonstrate that while reinforcement learning has made significant progress in automated penetration testing, several critical challenges remain. These include the need for more scalable algorithms capable of handling large and complex network environments, the issue of sparse rewards which can hinder effective learning, and the need for greater adaptability to dynamic network changes.

Future work should focus on addressing these challenges through innovations in algorithm design, integration of domain knowledge, and development

of more realistic simulation environments. By overcoming these obstacles, reinforcement learning has the potential to revolutionize the field of penetration testing, making it more efficient, effective, and accessible to a broader range of cybersecurity professionals.
Keywords: Reinforcement Learning, Automated Penetration Testing, Literature Review

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

With the increasing complexity and frequency of cyber attacks, traditional penetration testing methods are becoming increasingly inadequate. Penetration testing is a technique that simulates the behavior of an attacker to identify security vulnerabilities in a system. Traditional penetration testing often relies on manual operations, which are time-consuming and expensive, and the test results are highly dependent on the experience and skills of the testers [2]. To improve the efficiency and effectiveness of penetration testing, researchers are exploring the application of reinforcement learning in the field of penetration testing.

Reinforcement Learning is a branch of machine learning, that involves an agent interacting with the environment and continuously learning the optimal strategy to maximize cumulative rewards. In penetration testing, reinforcement learning can automate the process of discovering and exploiting system vulnerabilities, thereby improving the coverage and depth of the tests. In recent years, many research efforts have demonstrated the potential of reinforcement learning in automated penetration testing. For example, Schwartz et al. designed and built a network attack simulator that frames penetration testing as a Markov Decision Process (MDP) and uses reinforcement learning algorithms for training and testing, providing a basis for the use of reinforcement learning in penetration testing [1].

1.2 Objectives

To address the numerous challenges reinforcement learning faces in applying automated penetration testing, the academic community has devoted signif-

icant effort to researching and exploring more effective solutions. However, despite the progress in advancing this field, there remain several critical issues and shortcomings in existing research that limit the widespread application of reinforcement learning in automated penetration testing. Therefore, it is necessary to conduct a systematic review and comprehensive analysis of these studies. This will not only help in gaining a thorough understanding of the current research landscape but also in identifying hidden key issues and research gaps, which are crucial for guiding future work.

The main objectives of this study are:

1. **Systematic Literature Review**: Conduct a systematic literature review on the application of reinforcement learning in automated penetration testing to identify key challenges and gaps in current research.

2. **Evaluation Methods and Standards**: Design a set of reliable evaluation methods and standards to accurately assess the effectiveness of automated penetration testing.

3. **Experimental Validation**: Demonstrate the applicability and reliability of reinforcement learning agents in different network environments through experiments.

4. **Propose Future Research Directions**: Based on the research findings, propose specific future research directions to address key challenges in the current studies.

1.3 Originality and Significance

In recent years, reinforcement learning has made significant progress in the field of automated penetration testing. Nevertheless, this field still faces several challenges. For example, due to the training process relying mainly on simulators, models are difficult to apply directly to real network environments [1]. Additionally, as the action space expands, many studies struggle to scale effectively to large network environments [2]. Therefore, it is essential to systematically summarize and analyze the existing relevant research papers to gain a comprehensive understanding of the current state of research in this field. Although some review articles have explored the application of reinforcement learning in automated penetration testing, such as the comprehensive review of reinforcement learning applications in the field of network security by Cengiz and Gök [16], which covers penetration testing and Intrusion Detection Systems (IDS). However, the scope of this study is relatively

broad, involving multiple aspects of network security, and does not focus on automated penetration testing. Moreover, the number of studies related to automated penetration testing mentioned in the article is relatively limited. Similarly, in the research by Vyas and Hannay [17], the application of reinforcement learning in automated penetration testing is also involved, but this study also does not specifically focus on this particular field, and the number of relevant studies discussed is not large, with some of the latest research findings not included in their discussion scope. In addition, the research by Palmer and Parry [18] provides a comprehensive overview of the application of Deep Reinforcement Learning (DRL) in autonomous network defense, focusing on the challenges faced by high-dimensional state spaces, large combinatorial action spaces, and adversarial learning. This study is of high reference value, and compared with the aforementioned two studies, the number of relevant studies discussed has increased, but the scope of research is still relatively broad, and some of the latest research findings have not been covered, such as DynPen proposed by Qianyu Li and others [19].

This research will focus specifically on the application of reinforcement learning in automated penetration testing. The number of reviewed papers will reach around 30. Based on the literature analysis, we propose a systematic evaluation method and standards to assess the effectiveness of automated penetration testing. Furthermore, we conducted a series of experiments using NAS to ensure the reliability and credibility of the results.

The contributions of this study are as follows.

- **Comprehensive Analysis**: Conducted an in-depth analysis of approximately 30 relevant papers to identify key challenges, trends, and research gaps in the application of reinforcement learning for automated penetration testing.

- **Identification of Future Research Directions**: Through a comprehensive analysis of the papers, we identified key challenges in current research, such as scalability, sparse rewards, and dynamic adaptability, and proposed future research trends.

- **Experiments**: We conducted experiments using Network Attack Simulator (NAS)[1] to demonstrate the performance and reliability of reinforcement learning in automated penetration testing across different network scenarios.

- **Validation of Automated Penetration Testing Effectiveness**: Based on a comprehensive analysis of the papers and experimental results, we designed a set of feasible evaluation methods and standards to assess the effectiveness of automated penetration testing.

1.4 Report Structure

The remainder of this report is organized as follows:

- **Chapter 2** provides a detailed introduction to the fundamental knowledge related to this study.
- **Chapter 3** discusses the reviewed papers that applied the DQN algorithm as the reinforcement learning approach.
- **Chapter 4** summarizes studies that utilized other reinforcement learning algorithms or alternative methods.
- **Chapter 5** provides an overall analysis of all the reviewed papers.
- **Chapter 6** presents the experiments and results.
- **Chapter 7** concludes the report.

Chapter 2

Relevant Foundational Background Knowledge

2.1 Markov Decision Process (MDP)

The Markov Decision Process (MDP) model was first proposed by Richard Bellman in the 1950s during his research on Dynamic Programming. It was introduced to formalize and solve decision-making problems in stochastic environments. Through a sequence of decisions, the MDP framework addresses the challenge of achieving objectives in environments characterized by uncertainty and randomness. His research laid a crucial theoretical foundation for reinforcement learning (RL) and optimization theory.

In their research, Watkins and Dayan proposed the Q-Learning algorithm [20], which explicitly applied the MDP framework to solve reinforcement learning problems. The study used the MDP's states, actions, rewards, and transition dynamics to define optimization objectives for reinforcement learning. By introducing value function updates, they proposed an algorithm capable of learning optimal strategies in model-free environments.

The goal of reinforcement learning is to enable an agent to interact with its environment and, through trial-and-error learning, discover a policy (π) that maximizes long-term cumulative rewards. Reinforcement learning problems can typically be modeled as MDPs, with the following components:

- **State (S):** A representation of the environment at a specific moment.
- **Action (A):** The set of actions an agent can take in a given state.
- **Transition Probability (P):** The probability of transitioning from one state to another, which is often unknown.

- **Reward Function (R):** The immediate feedback signal provided by the environment after an action is taken.
- **Policy (π):** The behavioral rule of the agent, defining a probability distribution over actions for each state.

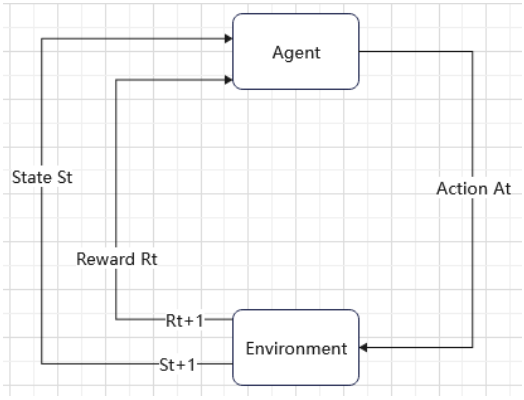


Figure 2.1: Schematic diagram of the Markov Decision Process model

The task of reinforcement learning is equivalent to solving for the optimal policy (π^*) in an MDP, which maximizes cumulative rewards over time.

2.2 Partially Observable Markov Decision Process (POMDP)

Partially Observable Markov Decision Process (POMDP) is a mathematical framework used for modeling decision-making problems. The theoretical foundation of POMDP can be traced back to the operations research and control theory of the 1960s. In 1973, R.A. Smallwood and E.J. Sondik formally proposed the mathematical framework of POMDP in their paper and studied the optimal control problem, making significant contributions to the development of POMDP [21]. In the 1990s, Leslie P. Kaelbling, Michael L. Littman, and Anthony R. Cassandra systematically explored the application of the POMDP framework in reinforcement learning through multiple studies [22, 23, 24]. Their research delved into how to learn policies in partially observable environments and proposed several early solution algorithms. POMDP extends the traditional MDP by introducing the concepts of observation and belief states, enabling it to address problems involving uncertainty and partial observability.

Definition of POMDP
A POMDP is defined as a tuple:

$$\langle S, A, T, R, \Omega, O, \gamma \rangle$$

- **State Set (S)**: A set of all possible states of the system, representing the true state of the environment. These states may not be directly observable by the decision-maker.
- **Action Set (A)**: A set of all possible actions that the decision-maker can choose from.
- **State Transition Function ($T(s' \mid s, a)$)**: Describes the probability distribution of transitioning to state s' after taking action a in state s.
- **Reward Function ($R(s, a)$)**: Specifies the immediate reward received when action a is taken in state s.
- **Observation Set (Ω)**: A set of all possible observations that the decision-maker might receive, which provide partial information about the environment's state.
- **Observation Function ($O(o \mid s', a)$)**: Describes the probability distribution of observing o after taking action a and transitioning to state s'.
- **Discount Factor ($\gamma \in [0, 1)$)**: Balances the importance of immediate rewards and long-term rewards. Smaller values emphasize short-term gains, while larger values prioritize long-term benefits.

Differences Between POMDP and MDP POMDP is an extension of MDP, and there are significant differences between the two in terms of model assumptions, state observability, and complexity.

1. **State Observability**
    - **MDP:** Assumes that the system's state is fully observable, meaning the agent can precisely know the current state.
    - **POMDP:** Assumes partial observability of the system's state. The agent cannot directly observe the true state but instead receives incomplete information through observations.
2. **State Representation**
    - **MDP:** States are directly represented as explicit state s.

- **POMDP:** States are often represented as a probability distribution known as the *belief state*, which estimates the agent's belief about the current state.

3. **Observation Model**

   - **MDP:** Does not require an observation model because the state is fully observable.
   - **POMDP:** Includes an observation model that describes how hidden true states generate observations. This is typically defined as $O(o \mid s', a)$, representing the probability of observing o after taking action a and transitioning to state s'.

4. **Complexity**

   - **MDP:** Relatively simple, as solving it only requires considering the impact of current states and actions.
   - **POMDP:** More complex, requiring the agent to optimize both action selection and belief state updates, resulting in significantly higher computational demands.

2.3 Network Attack Simulator (NAS)

In 2018, Jonathon Schwartz published a paper titled *"Automated Penetration Testing Using Reinforcement Learning"* [1]. According to the author, this was the first study to apply reinforcement learning (RL) techniques to automated penetration testing. This innovative idea of employing RL in automated penetration testing has since been widely adopted in subsequent studies. Furthermore, the author developed the Network Attack Simulator (NAS), which filled a gap in the field of automated penetration testing by providing an open-source and simple experimental platform for training automated penetration testing agents. The simulator has become a key experimental platform for subsequent research, significantly advancing cybersecurity automation. The source code (version 0.12.0) is publicly available at `https://github.com/Jjschwartz/NetworkAttackSimulator`. Additionally, accompanying documentation can be found at `https://networkattacksimulator.readthedocs.io/en/latest/`.

2.3.1 Research Goals and Contributions

The primary goal of this research is to tackle the inefficiencies of the current penetration testing process and the shortage of skilled cybersecurity profes-

sionals. Penetration testing is a key method for evaluating network security, but it typically requires a high level of expertise and is time-consuming. Automated penetration testing presents a promising solution that can maintain effective network security assessments while mitigating the reliance on specialized human resources.

This study comprises two main components:

1. **Development of the NAS:** NAS provides an experimental environment for evaluating automated penetration testing agents. It is a fast, lightweight, open-source solution designed to simulate networks in a simple yet flexible manner. By abstracting away the complexities of real networks, it enables automated agents to make efficient decisions without focusing on low-level details.

2. **Application of RL:** The author explored the feasibility of using RL to generate optimal attack strategies in a simulated environment. RL was chosen for its ability to solve Markov Decision Processes (MDPs) without requiring detailed environmental models. By learning optimal behavioral policies through interaction with the environment, RL is particularly advantageous in the dynamic field of cybersecurity.

2.3.2 Implementation of NAS

NAS consists of two main components. The first is the **network model,** which contains all the data structures and logic necessary to simulate network traffic and attacks. The network model includes multiple subnets, topologies, machines, services, and firewalls. Each subnet contains multiple machines that can fully communicate with one another, while communication between subnets is controlled by topology and firewall configurations.

Each machine in the network model is defined by its address, value, and configuration. The configuration includes services running on the machine, which may have vulnerabilities exploitable by attackers. Firewalls are used to control traffic between subnets and to define permissible service types.

The second component is the **environment,** built on the network model as an interface between the attacker and the network. The environment is modeled as an MDP to simulate the attacker's position and knowledge state within the network. The state space includes information about each machine in the network, such as whether it has been compromised, its reachability, and the status of its services. The action space includes scanning machines and exploiting service vulnerabilities. The reward function is calculated by subtracting the action cost from the value of compromised machines, aiming

to enable the RL agent to compromise high-value machines with minimal actions.

2.3.3 Reinforcement Learning Training Details

Training RL agents in NAS is achieved through repeated interactions with the environment. Specifically, the author used Q-learning and neural network-based RL algorithms. Q-learning, a value iteration method, incrementally learns the optimal policy by constructing a state-action value table (Q-table). Neural network-based algorithms leverage Deep Q-Networks (DQNs) to handle larger state spaces, improving the scalability of Q-learning.

During training, the RL agent initializes the environment and begins interacting from the initial state. At each step, the agent selects and executes an action based on the current state, obtaining a new state and corresponding reward. The agent updates its policy based on these experiences. To balance exploration and exploitation, an ε-greedy strategy is employed, allowing the agent to select random actions with a certain probability to explore new possibilities.

Experiments demonstrated that both tabular Q-learning and neural network-based RL algorithms could find effective attack paths in the simulated environment. However, scalability remains a significant challenge. These methods performed well in small-scale networks but faced substantial difficulties in larger networks due to the rapid growth of state and action spaces. Thus, the study emphasizes the need for future improvements in the scalability and performance of these algorithms. As shown in Figure 2.2, illustrates the architecture of NAS.

2.3.4 Experimental Results

Experimental results indicate that both Q-learning and neural network-based RL algorithms successfully identify optimal attack paths in networks of varying scales and configurations. However, scalability remains a major issue. While these methods excel in small-scale networks, the expansion of state and action spaces in larger networks leads to significant challenges.

The author notes that, despite the success of these RL methods in small-scale environments, further research is needed to improve their scalability and performance. Additionally, higher-fidelity environments are necessary to evaluate the effectiveness of these methods in real-world applications.

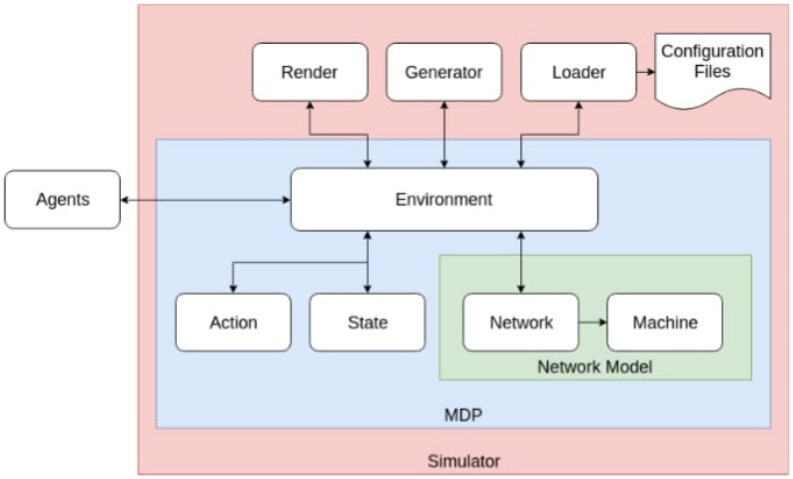


Figure 2.2: Network Attack Simulator program architecture [1].

2.3.5 Impact and Future Work

This research lays the foundation for applying RL to automated penetration testing. The NAS developed by the author has been widely adopted as a research platform, highlighting its importance in advancing cybersecurity automation. Future work includes developing scalable RL algorithms capable of handling larger networks and testing these algorithms in higher-fidelity environments to evaluate their potential for commercial applications.

2.4 CyberBattleSim

2.4.1 Research Background and Objectives

With the ever-evolving nature of cyber threats and the growing complexity of network infrastructures, traditional penetration testing methods face increasing challenges in terms of time and resource requirements. Automated penetration testing based on Reinforcement Learning (RL) has thus emerged as a promising approach, aiming to enhance testing efficiency and alleviate the shortage of skilled security professionals. The source code is publicly available at `https://github.com/microsoft/CyberBattleSim`.

Microsoft's **CyberBattleSim** is an open-source platform that leverages RL for network attack simulation, providing researchers and engineers with an environment to develop and evaluate automated penetration testing strategies [25]. Prior to CyberBattleSim, the **NetworkAttackSimulator (NAS)**, introduced by Jonathon Schwartz, laid the groundwork as one of the earliest

open-source simulators applying RL to automated penetration testing. This section summarizes the main features and research value of CyberBattleSim, while drawing comparisons with NAS.

2.4.2 System Design and Functionality

Network Modeling and Abstraction

CyberBattleSim represents the network as a collection of interconnected *nodes* (e.g., hosts, servers, containers) with links (edges) indicating reachability or trust relationships. Each node possesses various security properties, including vulnerability information, compromised states, and privilege levels. Through a flexible configuration interface, researchers can tailor the network topology, node attributes, and vulnerability distributions to represent different scales and complexities.

In a similar vein, NAS also employs an *abstracted network model*, featuring subnets, topologies, machines, services, and firewalls. Compared to NAS's lightweight abstraction, CyberBattleSim offers more extensive configuration of nodes and vulnerability details, enabling the simulation of more diverse and intricate attack scenarios.

Attacker–Environment Interaction

Like NAS, CyberBattleSim models the penetration testing process as a Markov Decision Process (MDP). The *state space* encapsulates the attacker's knowledge of the network (e.g., compromised nodes, reachable areas, vulnerability states), while the *action space* includes scanning, exploiting vulnerabilities, and lateral movement. The environment grants a *reward or penalty* for each action to guide the agent toward achieving higher-value nodes or efficient exploitation.

This design is closely aligned with NAS, where an RL agent interacts with a simulated environment, learning an optimal attack policy through iterative exploration and exploitation.

Visualization and Scalability

CyberBattleSim not only provides a Python-based API but also integrates visualization components to track and analyze the attack process. By leveraging Microsoft Azure and other cloud services, users can deploy large-scale experiments and automate security testing pipelines.

NAS similarly emphasizes open-source accessibility and ease of use but remains more suitable for small-scale experiments. For extensive or highly com-

plex network simulations, additional optimizations or customizations might be required in NAS to maintain performance, whereas CyberBattleSim benefits from stronger integration with Microsoft's cloud ecosystem.

2.4.3 Reinforcement Learning Training Details

In CyberBattleSim, RL algorithms such as Q-learning and Deep Q-Networks (DQN) can be employed for training automated attack strategies. The training procedure typically includes:

1. **Environment Initialization:** Load a predefined network topology, node information, and vulnerability distribution.
2. **Agent–Environment Interaction:** At each timestep, the agent chooses an action (e.g., scan or exploit) based on its current state, obtains immediate rewards, and transitions to a new state.
3. **Policy Update:** Value iteration or deep RL methods update the policy to gradually approximate an optimal or near-optimal attack strategy.
4. **Exploration–Exploitation Balance:** An ϵ-greedy approach or similar method ensures sufficient exploration of unseen states while utilizing learned experience.

Likewise, NAS implements tabular Q-learning and DQN to illustrate how RL can effectively discover efficient attack paths in smaller networks. Both frameworks, however, face the major obstacle of *state-action space explosion* when the network grows in scale or complexity.

2.4.4 Comparison with NAS

- **Open Source Ecosystem:** Both CyberBattleSim and NAS are open source, featuring documentation, code samples, and community support. NAS has been widely adopted in academic contexts, whereas CyberBattleSim, backed by Microsoft's cloud ecosystem, exhibits strong potential for enterprise deployment.
- **Simulation Details and Flexibility:** NAS offers a streamlined, highly abstract model suited for rapid prototyping; CyberBattleSim provides richer customization options, beneficial for modeling complex attack scenarios.

- **Scalability:** NAS is notably easy to configure for small-scale networks; CyberBattleSim, with Azure integration, can be scaled up for larger and more complex network simulations.
- **Visualization and Toolchain Integration:** While NAS focuses on the simulator core and basic logging, CyberBattleSim includes advanced visualization and analytics, and can integrate with broader Microsoft security services.

CyberBattleSim and **NAS** both exemplify how reinforcement learning can be harnessed for automated penetration testing. They share the core approach of training RL agents to discover optimal or near-optimal attack strategies within a simulated environment, thereby lowering labor costs and improving penetration testing efficacy. CyberBattleSim, with Microsoft's cloud infrastructure and expanded customization features, is poised for broader, more complex use cases, while NAS remains a simpler yet influential platform central to academic research.

2.5 MulVAL

In 2005, **Xinming Ou** and his colleagues presented a paper titled *"MulVAL: A Logic-based Network Security Analyzer"* [26] at the USENIX Security Symposium. According to the authors, this was the first time logical reasoning techniques were systematically applied to develop tools for multi-host, multi-stage vulnerability analysis. This innovative research has since been widely adopted in subsequent studies and practices, becoming a foundational contribution to the field of automated network security analysis. The MulVAL platform, developed by the authors, introduced the automated generation and reasoning of attack graphs, filling a critical gap in traditional security evaluations by providing an efficient and cost-effective tool to help network administrators identify potential threats, optimize security strategies, and evaluate the impact of multi-stage attacks. The platform is open-source, with the source code available at `https://github.com/risksense/MulVAL`, and supporting documentation at `https://MulVAL-docs.readthedocs.io/`.

2.5.1 Research Objectives and Contributions

The primary goal of this research is to address the inefficiency of manual evaluations for complex attack paths in network security analysis and the lack of systematic tools to support such analyses. Traditional vulnerability assessments and security evaluations often rely on human expertise and reasoning,

which not only require extensive security knowledge and practical experience but also risk overlooking potential threats, especially in multi-stage, multi-host network environments. Therefore, MulVAL aims to combine logical reasoning with automated tools to offer an efficient, low-cost solution for analyzing multi-stage attack paths in complex network environments, enabling security professionals to quickly identify vulnerabilities and optimize defense strategies.

This study includes two main components:

1. **Logical Reasoning and Attack Graph Generation**: By leveraging Attack Graphs as a logic programming language, the authors abstracted network elements such as hosts, services, vulnerabilities, network topology, and firewall rules into a series of logical rules. Using a reasoning engine, these rules are automatically processed to deduce possible attacker actions and generate attack graphs, which visually represent all possible paths from an attack's starting point to its ultimate targets. This approach provides network administrators with intuitive threat analysis results, reducing cognitive overhead.

2. **Security Policy Evaluation**: By analyzing the generated attack graphs, researchers can systematically assess the effectiveness of existing security policies, identify hidden weaknesses in the network, and implement targeted defense measures to improve overall security.

2.5.2 Implementation Details of MulVAL

The implementation of MulVAL comprises two major components: the construction of the network model and the application of logical reasoning, both aimed at automating the analysis and generation of multi-stage attack paths:

1. **Network Model Construction**: The network model describes key elements in the network, including hosts, services, vulnerabilities, topology, and firewall rules. Each host is defined by its IP address, the services it runs, potential vulnerabilities, and relative value. Firewall rules govern communication between hosts. All this information is abstracted into logical rules as input for the reasoning engine, allowing complex network environments to be simplified and structurally represented.

2. **Logical Reasoning and Attack Graph Generation**: By employing Attack Graphs rules, the reasoning engine in MulVAL deduces all possible attack paths based on the network model and generates attack

graphs. These graphs not only display potential intrusion methods but also reveal vulnerabilities in defense strategies, offering comprehensive threat analysis results to administrators.

2.5.3 Experimental Results

Experimental results demonstrate that MulVAL efficiently generates detailed attack graphs in small and medium-sized network environments, showcasing its precision and practicality in analyzing multi-stage attack paths and identifying potential vulnerabilities. However, as the network scale increases, the state space of the logical reasoning engine expands rapidly, presenting significant challenges to its performance and scalability. This limitation highlights the need for future research to optimize the underlying algorithms and enhance the tool's capabilities for handling large-scale networks.

Nonetheless, MulVAL has proven to be a powerful support tool for network administrators, providing detailed threat analysis and helping to identify weaknesses in existing defense strategies, which serves as a practical basis for further research and the optimization of security policies.

2.5.4 Impact

The introduction of MulVAL has laid a critical foundation for the field of network security analysis. By combining logical reasoning with attack graph generation, it offers an efficient and intuitive solution for analyzing multi-host, multi-stage attack paths. Its methodologies have been widely adopted in subsequent research and practice, becoming a key tool in advancing automated network security analysis.

Chapter 3

Systematic Literature Review Related to DQN

Starting from this chapter, we will systematically review the literature related to the application of reinforcement learning in automated penetration testing. The objective is to classify and analyze the contributions, methodologies, and advancements of various studies. In this chapter, we first focus on research that employs DQN-based algorithms as their core reinforcement learning approach. These studies demonstrate how DQN is utilized, adapted, and extended to address the various complex challenges encountered in automated penetration testing.

The DQN algorithm was introduced in the paper *"Playing Atari with Deep Reinforcement Learning"* [27]. It combines deep learning with Q-learning to directly learn optimal policies from high-dimensional inputs, such as pixel images from games. It uses a deep convolutional neural network to estimate the state-action value function (Q-value) and incorporates two key techniques: Experience Replay and Target Network, to improve training stability. Experience Replay breaks the temporal correlation by randomly sampling stored experiences, while the Target Network avoids instability in the Q-value updating process by using delayed updates. Additionally, DQN employs an ε-greedy strategy to balance exploration and exploitation, enabling the algorithm to surpass human performance in several Atari games. DQN has been widely applied in the field of automated penetration testing due to its ability to learn and optimize attack strategies in complex environments automatically.

3.1 NIG-AP

In 2019, Tianyang Zhou et al. published a paper titled *"NIG-AP: a new method for automated penetration testing"* [28], proposing a method called Network Information Gain-based Automated Attack Planning (NIG-AP) for automating penetration testing. It is among the early studies to apply reinforcement learning to automated penetration testing.

3.1.1 Research Methodology

This study formalizes penetration testing as a Markov Decision Process (MDP) and employs a reinforcement learning model to guide the discovery of attack paths. Specifically, the NIG-AP algorithm is based on the concept of network information gain, treating each step in penetration testing as a process of reducing information entropy, thereby directing the choice of attack paths.

1. **Network Information Gain (ΔH)**:
    - Information gain represents the reduction in uncertainty of target network information after an attack operation.
    - In penetration testing, the state of each host is defined by information such as the operating system, installed software, open ports, and protection mechanisms, with information entropy describing the uncertainty in the host's exposure status.
    - By incentivizing attack actions based on information gain, the algorithm prioritizes operations that maximize information acquisition. A key advantage of this method is that penetration testing agents can progressively reduce uncertainty by continually gathering information, even without complete knowledge of the network structure, thus enabling effective attacks.
2. **Action Cost ($r_{\mathbf{cost}}$)**: Action cost is calculated using the Common Vulnerability Scoring System (CVSS) to limit unnecessary actions and optimize attack paths.
3. **Application of a Reinforcement Learning Model:** The study adopts the Deep Q-Network (DQN) algorithm, using information gain as a reward signal to guide penetration testing agents in intelligently selecting optimal attack actions.

3.1.2 Experiments and Results

Experimental Setup: The experiments were conducted in a typical enterprise network environment, including a DMZ area and an internal network connected via a firewall and a router. The experimental setup was fully virtualized to simulate real-world network environments, and Nessus was used to collect network information. The NIG-AP algorithm was compared with traditional methods, such as the Partially Observable Markov Decision Process (POMDP) [29] and Forward Search (FF) [30], to evaluate its performance in complex network scenarios.

Experimental Results:

- The NIG-AP algorithm achieved convergence precision comparable to POMDP after the fourth iteration, with significantly reduced training time.
- In identifying effective attack paths, NIG-AP demonstrated significantly higher effectiveness than POMDP and FF.
  - FF, unable to handle uncertainty, can only find a single attack path by exhaustive iteration, whereas NIG-AP identifies more potential paths.
- As the number of hosts increased, NIG-AP required much less training time than POMDP. Notably, when the number of hosts exceeded three, POMDP became infeasible for solving the problem.

3.2 AutoPentest-DRL

In 2020, Zhenguo Hu published a paper titled *"Automated Penetration Testing Using Deep Reinforcement Learning"* [2]. This paper proposed an automated penetration testing framework based on deep reinforcement learning (DRL), named AutoPentest-DRL. The aim is to simplify the penetration testing process through automation, reducing reliance on manual operations while improving efficiency and accuracy. The platform is released as open-source, with the source code available at `https://github.com/crond-jaist/AutoPentest-DRL`.

3.2.1 Framework Overview

The working principle of AutoPentest-DRL can be divided into two main stages: preprocessing of training data and DQN decision engine training.

Preprocessing of Training Data:

- **Data Collection:** The Shodan engine collects device information and vulnerability data from real network environments to ensure realistic training data.
- **Attack Tree Generation:** The MulVAL tool generates an attack tree for the network, mapping all possible attack paths within the environment.
- **Attack Path Matrix Creation:** The DFS algorithm traverses the attack tree, identifying all possible attack paths and creating a matrix that encodes potential attack steps and their corresponding reward values.

This matrix serves as the foundation for training the deep reinforcement learning algorithm.

Detailed Explanation of the Reward Mechanism The author defined a reward matrix R to initialize and update the matrix Q. This matrix contains the reward values for each attack step. These reward values are calculated based on the Common Vulnerability Scoring System (CVSS), as shown in the formula 3.1 [2]:

$$\text{Score}_{\text{vul}} = \text{baseScore} \times \frac{\text{exploitabilityScore}}{10} \tag{3.1}$$

This formula computes the reward value for each vulnerability, where **baseScore** represents the base severity score of the vulnerability, and the value **exploitabilityScore** represents the exploitability score of the vulnerability.

DQN Decision Engine Training: The simplified attack matrix is input into the DQN (Deep Q-Network) decision engine. During the training process, the agent in the DQN decision engine represents a real attacker, and the simplified matrix serves as the environment. Regardless of the values within the matrix, the agent can freely move within it. The reward matrix R is used to initialize and update the matrix Q, where each element $Q(s, a)$ in the Q-matrix represents the expected reward for taking the action a in state s.

Using the Q-learning algorithm, the DQN model continuously updates the Q-matrix to find the action policy that maximizes cumulative rewards. The ultimate goal is to reach the desired node, which indicates that the target has been successfully located. The output is a path that maximizes the cumulative rewards, representing the most effective attack strategy. The architecture of AutoPentest-DRL as shown in Figure 3.1.

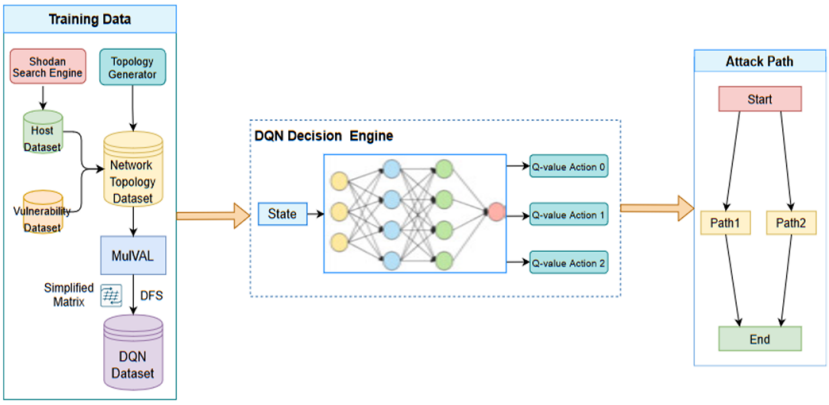


Figure 3.1: Architecture of the Autopentest-DRL [2].

3.2.2 Experimental Validation

The effectiveness of AutoPentest-DRL is validated through experiments in two scenarios: logical network testing and real network environment testing.

Logical Network Testing:

- Two network topology models are constructed: a simple home network topology and a more complex enterprise network topology.
- Each node in the logical network is initialized with device information, vulnerabilities, and open port data. The MulVAL tool generates attack trees for these networks.
- The DQN decision engine uses matrix input to identify the optimal attack path.

During this process, hyperparameters like the discount factor and batch size are adjusted to optimize DQN performance. Results show that AutoPentest-DRL quickly converges and identifies the attack path with the highest cumulative reward, demonstrating its feasibility in logical networks.

Real Network Environment Testing:

- A simulated enterprise network scenario is created using virtual machines.
- The framework uses Nmap to scan devices and obtain real-time vulnerability information, while Metasploit executes automated attacks.
- Multiple vulnerabilities are successfully exploited to gain root access to the target server, and a test Trojan is uploaded to the target device.

These results demonstrate that AutoPentest-DRL can efficiently conduct penetration testing in real-world environments, even in complex network architectures.

3.3 ASAP

In 2020, Ankur Chowdhary and his team published a paper titled *"Autonomous Security Analysis and Penetration Testing"* [3], proposing an automated penetration testing framework called ASAP. The authors have made the source code publicly available on GitHub, as follows: `https://github.com/ankur8931/asap`.

The overall architecture of the ASAP framework consists of several modules that form a complete automated penetration testing process:

1. **Network Scanning Module**: Collects vulnerability information from the target network.
2. **Attack Graph Generation Module**: Creates attack graphs based on collected information to describe relationships between different vulnerabilities.
3. **State Graph Generation Module**: Parses attack graphs to generate state and reward parameters required for reinforcement learning models.
4. **Deep Reinforcement Learning Module**: Uses the DQN algorithm to generate optimal penetration paths.
5. **Attack Verification Module**: Validates the feasibility of these paths in real environments using tools like Metasploit.

3.3.1 Attack Graph Generation and State Representation

First, tools such as Nessus and OpenVAS are used to scan the target network, collecting data on vulnerabilities (including CVSS scores and access complexity), network topology, and host configurations. This data is then fed into the MulVAL tool to generate attack graphs.

An attack graph is a directed graph where nodes represent the attacker's privilege state or vulnerabilities in the network, and edges represent exploitation paths for these vulnerabilities. By further analyzing the attack graph,

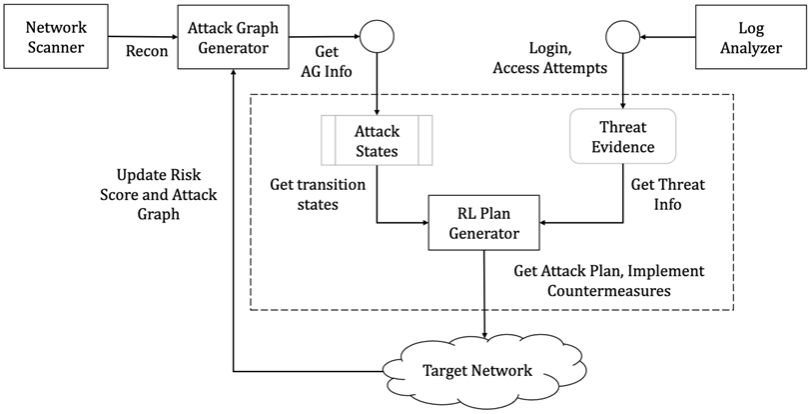


Figure 3.2: ASAP architecture and data flow [3].

the authors extracted input parameters required for reinforcement learning, including state space, action space, reward function, and state transition probabilities.

The transition probability of each edge in the attack graph is determined by the vulnerability's access complexity, classified into three categories:

- **LOW**: Success probability 0.9.
- **MEDIUM**: Success probability 0.6.
- **HIGH**: Success probability 0.3.

3.3.2 Deep Reinforcement Learning for Generating Optimal Attack Paths

ASAP uses a Deep Q-Network (DQN) to build its reinforcement learning model. In this model, each state-action pair represents a specific privilege escalation step, and the reward system is quantified based on the Common Vulnerability Scoring System (CVSS) score, with a maximum score of 10. Vulnerabilities with higher severity receive higher rewards, guiding the reinforcement learning algorithm to prioritize these vulnerabilities.

ASAP also incorporates a domain-specific state transition probability model to reflect the difficulty of exploiting different vulnerabilities—some vulnerabilities have lower success probabilities, which directly influence the selection of attack paths. This modeling enables ASAP's deep reinforcement learning model to effectively handle the complexity of network environments and generate more targeted attack strategies.

As shown in Figure 3.2, the architecture and data flow of the ASAP framework illustrate the interaction processes between various modules. Scanning information is passed to the attack graph generator, and the core threat analysis component based on reinforcement learning utilizes the attack graph and log information to create an attack plan.

3.3.3 Validation of ASAP Framework

The effectiveness of the ASAP framework was validated through experiments:

- **Enterprise Network Case Study** ASAP was deployed in a simulated enterprise network environment consisting of 16 hosts and multiple subnets, including Industrial Control Systems (ICS) and Internet of Things (IoT) devices, running several critical services (e.g., SSH, HTTP, SMTP, and FTP). The experiments demonstrated that ASAP could not only converge quickly to find the optimal attack path but also uncover hidden attack routes in complex network environments.
- **Large-Scale Network Testing** To evaluate its scalability in complex scenarios, ASAP was tested on a large-scale network with 300 hosts. In this environment, ASAP was able to generate the optimal attack path in approximately 70 seconds.

3.3.4 Future Directions

The authors proposed extending ASAP to more scenarios in the future, such as addressing Advanced Persistent Threats (APT) and supporting red-blue team exercises (also known as purple teaming). With these extensions, ASAP will be better equipped to handle complex attack scenarios, further enhancing its diversity and practicality in the field of information security.

3.4 NDSPI-DQN

In 2021, Shicheng Zhou et al. published a paper titled *"Autonomous Penetration Testing Based on Improved Deep Q-Network"* [4]. This paper proposed an enhanced autonomous penetration testing framework that utilizes an improved DQN algorithm, named NDSPI-DQN, designed to address two major challenges in large-scale network scenarios: the sparse reward problem and the large action space.

3.4.1 Enhancements to the DQN Algorithm

The framework consists of two main parts. First, the DQN algorithm is improved through the following five enhancements:

- **Noisy Nets:** By adding Gaussian noise to the network layers, the action selection process gains stochasticity. Compared to the traditional ϵ-greedy strategy, this allows for exploring more potential paths.
- **Soft Q-learning:** Incorporating the principle of Maximum Entropy, entropy rewards are added to the action selection process to generate more diverse strategies. The action selection strategy employs a softmax approach, assigning probability weights to each action instead of solely choosing the greedy action.
- **Dueling Architectures:** The Q-value function is decomposed into a Value Function and an Advantage Function. The Value Function evaluates the importance of the current state, while the Advantage Function assesses the relative quality of specific actions in that state. This structure enables more efficient evaluation of action selection in complex network environments.
- **Prioritized Experience Replay:** Experiences are prioritized based on Temporal Difference Error (TD Error), with higher priority given to experiences that have a significant impact on the current policy. This enhances the effectiveness of experience replay and accelerates training convergence.
- **Intrinsic Curiosity Model (ICM):** An intrinsic reward signal is used to guide the agent to explore environments with sparse external rewards by predicting the next state. The intrinsic reward, driven by the model's prediction error, encourages the agent to explore unfamiliar states. This curiosity-driven exploration generates internal rewards, addressing the sparse reward problem.

3.4.2 Framework Architecture

As shown in Figure 3.3, the framework's learning cycle and overall NDSPI-DQN architecture are composed of two parts:

- **Learning Cycle (Top):** The agent observes the current environment state s_t, outputs an action a_t based on the learned policy, and receives

an external reward r_{te} from the environment. Simultaneously, the Intrinsic Curiosity Model (ICM) takes (s_t, a_t, s_{t+1}) as input and outputs an intrinsic reward r_{it}. Both external and intrinsic rewards are used to optimize the policy.

- **NDSPI-DQN Architecture (Bottom):** Orange layers represent fully connected layers, and red layers denote dueling network architectures with noisy linear layers. The input to the network is the environment state vector, and the output is the action value.

These enhancements ensure that the reinforcement learning agent can efficiently explore large-scale network environments and learn optimal attack paths without requiring complete prior knowledge of the system.

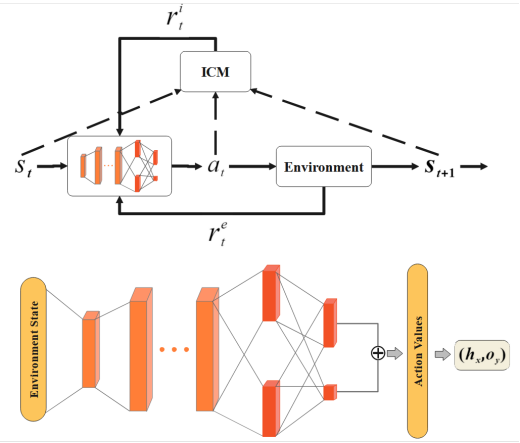


Figure 3.3: Learning cycle (top) and schematic illustration of NDSPI-DQN (bottom) [4].

3.4.3 Action Space Reduction

To reduce the action space, the NDSPI-DQN algorithm decomposes actions into two parts: the value evaluation of target hosts and the value evaluation of operation types. This allows the agent to estimate the values of target hosts and operation types separately, then combine them into a complete action. This decoupling strategy reduces the action space from $O(M \times N)$ **to** $O(M + N)$ (where M is the number of hosts and N is the number of executable operations), significantly reducing exploration and computation costs. This transformation reduces complexity from exponential to linear, greatly improving the algorithm's convergence speed in large-scale networks.

As shown in Figure 3.4, the decoupled NDSPI-DQN uses two streams that share hidden layers, with each stream outputting its respective Q-value through a dueling network architecture with noisy linear layers. This design effectively reduces the complexity of the action space while maintaining high performance.

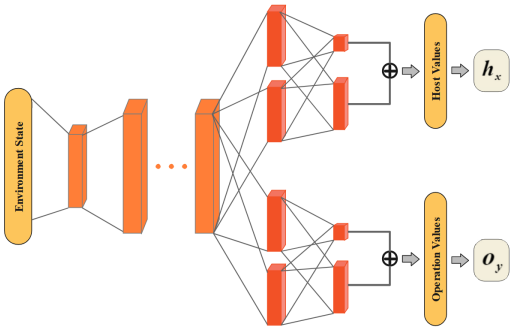


Figure 3.4: Schematic illustration of the decoupling NDSPI-DQN [4].

3.4.4 Experimental Evaluation

The experimental evaluation consisted of two phases:

Performance Comparison in Small-Scale Networks

In the first phase, the framework was tested in a standard network scenario containing 17 hosts to compare the performance of the improved DQN with the traditional DQN. Results showed that NDSPI-DQN significantly outperformed the original algorithm, converging faster and more efficiently.

Scalability Analysis in Large-Scale Networks

In the second phase, the framework was tested in larger network scenarios, with the network size reaching up to 150 hosts. The experimental results demonstrated that the decoupling mechanism allowed NDSPI-DQN to maintain stable convergence even as the network scale increased. This scalability and robustness make the framework particularly effective in handling large and complex networks where traditional penetration testing methods struggle due to the increased state and action space.

3.4.5 Conclusion and Future Work

The framework's successful validation in complex network scenarios highlights its potential for real-world applications. However, the current research is limited to simulated environments, and further studies are required to apply these methods to real networks.

3.5 CJA-RL

In 2021, Rohit Gangupantulu published a paper titled *"Crown Jewels Analysis using Reinforcement Learning"* [5]. This paper proposed a methodology called CJA-RL (Crown Jewel Analysis using Reinforcement Learning). Crown Jewel refers to critical IT assets.

3.5.1 Overview of CJA-RL

The core working principle of CJA-RL is based on leveraging reinforcement learning to probe the network structure and analyze potential attack paths. This method effectively identifies critical entry points, choke points, and pivot nodes within the network, which attackers could exploit to compromise the Crown Jewels.

The process is divided into two main stages:

- Understanding the network's structure and generating an attack graph.
- Applying reinforcement learning to simulate penetration testing and determine the optimal attack paths.

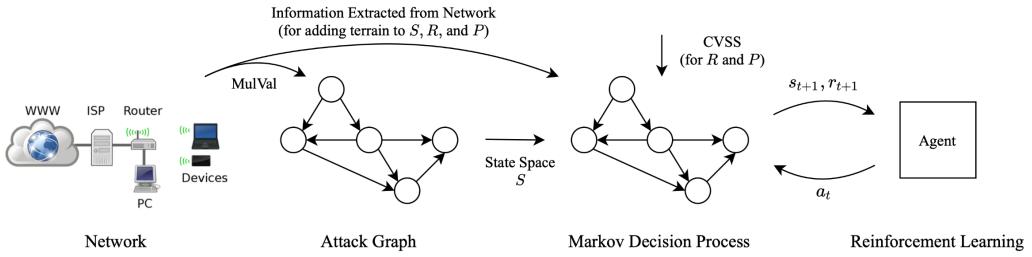


Figure 3.5: Deploying RL on MDPs modeled using attack graph representations of networks [5].

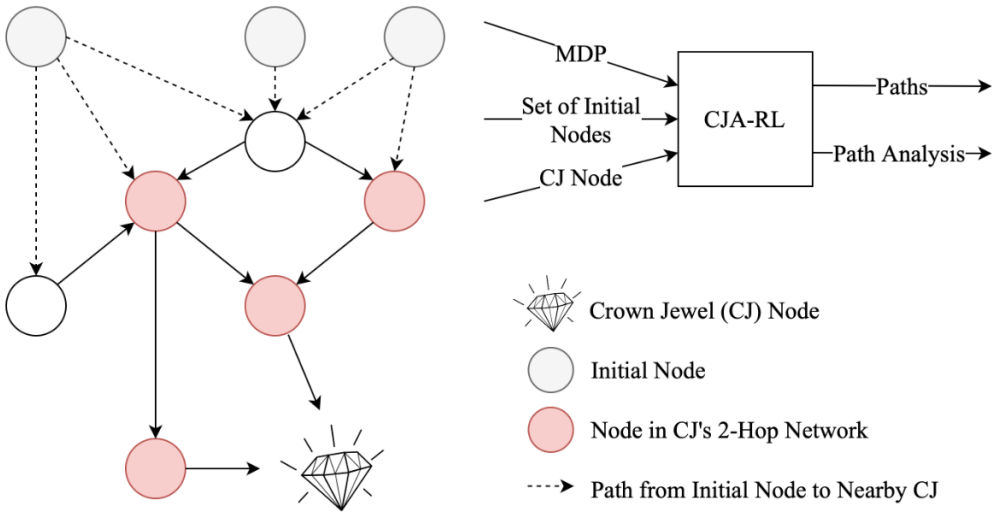


Figure 3.6: Depiction of CJA-RL [5].

3.5.2 Attack Graph Generation

In the first stage, the MulVAL tool is used to generate an attack graph that describes the network configuration, vulnerabilities, and possible attack paths. This attack graph forms the basis for the Markov Decision Process (MDP), where:

- Nodes represent components within the network.
- Edges represent potential attack actions.

The Common Vulnerability Scoring System (CVSS) is employed to quantify the severity of these vulnerabilities, and transition probabilities for different attack actions are assigned based on the complexity of exploiting each vulnerability. On this basis, the reinforcement learning algorithm (Deep Q-Network, DQN) is trained to learn how to find the optimal attack path from any initial node in the network to the Crown Jewels.

3.5.3 Reinforcement Learning for Attack Path Discovery

In the second stage, CJA-RL trains reinforcement learning agents to automatically discover the optimal path from initial nodes to target Crown Jewels. This approach not only evaluates the effectiveness of the attack paths but also considers the "quietness" of the attack—how effectively vulnerabilities can be exploited without triggering defensive systems in the network.

By simulating multiple attack behaviors, CJA-RL analyzes:

- Pivot and choke points along the attack paths.
- Key locations for lateral movement within the network.

These points are crucial for attackers and serve as important areas where defenders can deploy monitoring or countermeasures.

3.5.4 Experimental Evaluation and Results

To validate the effectiveness of CJA-RL, the paper conducted experiments on a large-scale network, analyzing several Crown Jewels and evaluating:

- The best attack paths.
- The most strategic initial nodes.
- Crucial pivot points.

The results showed that CJA-RL could effectively identify critical terrains within the network, such as quiet entry points, pivotal nodes, and optimal paths for attacking Crown Jewels. Additionally, the method was able to detect crucial choke points that attackers might use to traverse the network.

This information is valuable for network administrators to deploy effective defensive strategies and prevent potential attacks.

3.6 HA-DRL

Khuong Tran and his team proposed a groundbreaking deep reinforcement learning (DRL) architecture in their 2021 paper *"Deep Hierarchical Reinforcement Agents for Automated Penetration Testing"* [6], called HA-DRL (Hierarchical Agent Deep Reinforcement Learning), for automated penetration testing. The primary objective of this research is to address the issue of excessively large discrete action spaces in penetration testing. Traditional DQN methods typically require evaluating all possible actions to find the optimal one. However, in penetration testing, the action space can grow exponentially due to the increasing number of hosts, subnets, and potential vulnerabilities.

3.6.1 Challenges in Large Action Spaces

One of the biggest challenges in penetration testing with reinforcement learning is the exponential growth of the action space. Traditional DQN methods require evaluating all possible actions in each state, making it computationally infeasible when dealing with large networks. As the number of hosts, subnets, and vulnerabilities increases, the number of possible actions expands dramatically, leading to slow convergence and poor scalability.

3.6.2 Hierarchical Agent-Based Approach

To solve this issue, the HA-DRL architecture introduces the concepts of hierarchical agents and action space decomposition, instead of relying on a single agent to explore all possible actions. Specifically, HA-DRL:

- Divides the action space into smaller subspaces.
- Assigns a dedicated reinforcement learning agent to each subspace.

Through this hierarchical structure, different agents handle different dimensions of the actions. For example:

- One agent may focus on selecting which host to attack.
- Another agent decides the specific attack method.

This decomposition of the action space significantly reduces the number of actions each agent needs to process, resulting in more stable learning and faster convergence. Each agent independently learns the optimal policy for its respective subspace while sharing the same reward signal. This allows multiple agents to work in parallel, simultaneously exploring their respective action spaces and avoiding the bottleneck of a single agent exploring the entire space.

3.6.3 Hierarchical Action Selection

The final action is derived by combining the outputs of these hierarchical agents. This hierarchical action selection approach:

- Improves efficiency.
- Enhances the algorithm's performance in complex network scenarios.
- Enables faster identification of optimal attack paths.

Particularly in cases with thousands of possible actions, where traditional DQN struggles to converge, HA-DRL demonstrates stronger learning capabilities and better adaptability to large-scale networks.

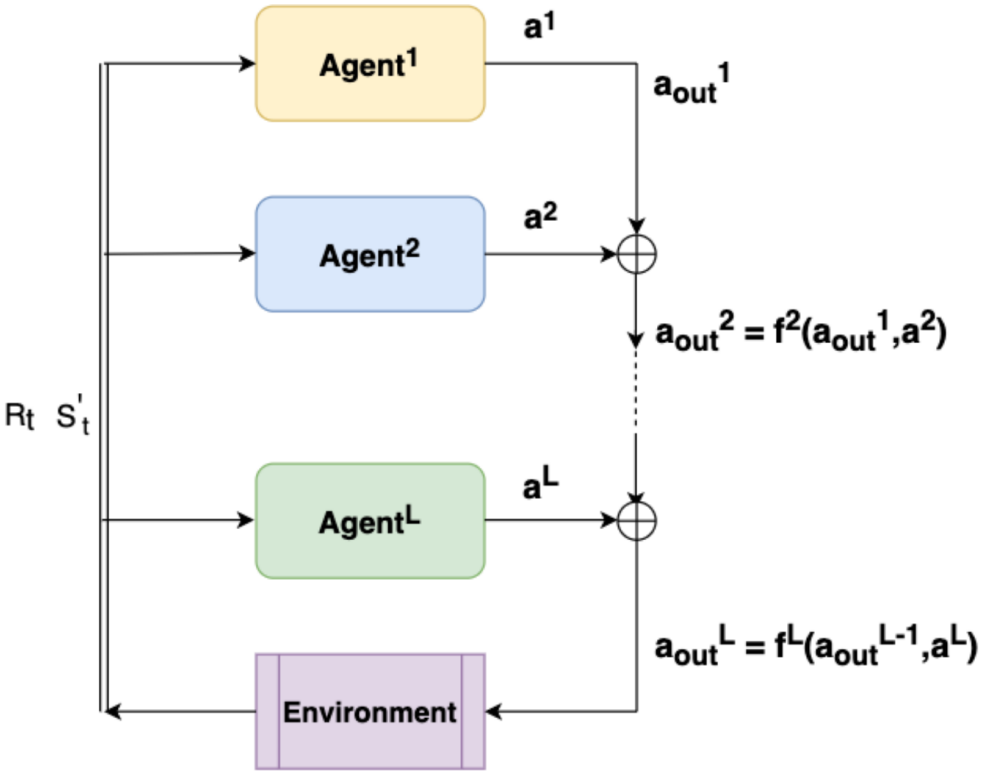


Figure 3.7: Architecture of HA-DRL [6].

3.6.4 Experimental Evaluation

The paper validates the HA-DRL framework through experiments using the CybORG simulator. The experiments covered various network configurations, ranging from:

- 6 to 100 hosts.
- Action spaces as large as 4646 actions.

The results show that HA-DRL converges faster and more reliably compared to a standard DQN agent, particularly in more complex network scenarios. In large action spaces, where traditional DQN struggles to explore

effectively, HA-DRL's hierarchical agent structure allows it to handle these complex scenarios with greater stability and success.

3.6.5 Interpretability and Scalability

Furthermore, the authors explore the interpretability of the action policies learned by HA-DRL, demonstrating how hierarchical agents efficiently group actions in complex networks. Each agent learns an optimal policy for its subspace, and the combination of these policies results in a global optimal strategy.

This architecture provides a scalable solution for addressing the challenges posed by large action spaces in penetration testing. As network sizes and action complexities continue to grow, HA-DRL offers an efficient framework for automated penetration testing.

3.6.6 Future Research Directions

The research indicates that HA-DRL has great potential for automating penetration testing. The authors suggest that future research could further optimize performance in sparse reward environments by:

- Enhancing exploration strategies.
- Incorporating subgoal learning.

These improvements could further increase the efficiency and applicability of HA-DRL in real-world cybersecurity scenarios.

3.7 CRLA

In 2022, Khuong Tran and his colleagues published a paper titled *"Cascaded Reinforcement Learning Agents for Large Action Spaces in Autonomous Penetration Testing"* [31]. This paper proposed a novel reinforcement learning framework called Cascaded Reinforcement Learning Agents (CRLA).

3.7.1 Overview

The authors described CRLA as a cascaded multi-agent architecture that organizes multiple reinforcement learning agents in a hierarchical structure, where each agent is responsible for a different subset of the entire action space. By algebraically decomposing the action space, each agent only needs

to learn within a smaller subset, significantly reducing learning complexity and computational overhead. However, after reviewing the architecture of this cascaded multi-agent framework and the hierarchical agent architecture (HA-DRL) proposed by the same authors in 2021, there appears to be no substantive difference between them. The segmented tree-based action organization method proposed in CRLA is also identical to the one in HA-DRL. Therefore, the improvements of CRLA over HA-DRL are as follows:

1. **Use of Dueling DQN**
   CRLA employs Dueling Deep Q-Networks (Dueling DQN) to train each agent independently. Dueling DQN is a variant of reinforcement learning that separates the state value and action advantage, making it more efficient in estimating the value of each state. This helps agents find the optimal strategy more quickly and stably. In this architecture, all agents share the state and reward signals from the environment but output different action components to collectively accomplish the entire attack process.

2. **Introduction of QMIX Mixing Network**
   CRLA introduces the QMIX mixing network mechanism, which aggregates the Q-values of each agent through a mixing network (MixingNet) to learn a global action value function. The global Q-value function guides all agents to collaborate for optimal decision-making, ensuring that each agent's behavior complements the others, ultimately achieving a globally optimal attack strategy. QMIX also adopts a hypernetwork structure to dynamically generate parameters for the mixing network, enabling CRLA to flexibly adapt to networks of varying sizes and complexities, further enhancing learning efficiency and stability.

3.7.2 Experiments

To validate the effectiveness of CRLA, the authors conducted experiments in two different environments.

Maze Environment They first tested CRLA's performance in handling large action spaces in a simplified maze environment. In this environment, the CRLA framework demonstrated outstanding performance, efficiently converging to the optimal strategy among 4096 possible actions. It performed particularly well in sparse reward scenarios, surpassing traditional single-agent DQN methods. Additionally, CRLA exhibited greater stability and faster convergence in action selection, showcasing its potential for similar applications.

CybORG Simulator The second set of experiments applied CRLA to penetration testing using the CybORG simulator. CybORG is a simulation platform designed for research in network security and automated penetration testing, capable of simulating real network environments with complex topologies. In the CybORG experiments, the authors tested CRLA in network scenarios of varying scales (up to 100 hosts) and action space complexities (up to 4646 actions), verifying its feasibility in practical applications. The results demonstrated that CRLA effectively identified optimal attack paths in all tested scenarios, even in very large and complex networks.

3.7.3 Comparison with HA-DRL

I compared the experimental results of CRLA with those of HA-DRL. The network scale and scenarios for both methods are identical (with a maximum of 100 hosts and 4646 actions). While CRLA introduces certain optimizations, it can only be said that CRLA is theoretically better suited for larger-scale networks and action spaces.

3.8 SmartGrid-PTDRL

Despite the effectiveness of traditional penetration testing (PT) methods in identifying vulnerabilities in software, hardware, and local area networks, these methods are insufficient for large-scale cyber-physical systems such as smart grids due to their complexity, coupling, and diversity. To address this challenge, Yuanliang Li and his colleagues proposed a DRL-based penetration testing framework in 2022, titled *"Deep Reinforcement Learning for Penetration Testing of Cyber-Physical Attacks in the Smart Grid"* [7], to efficiently identify critical vulnerabilities in smart grids. We denote this framework SmartGrid-PTDRL.

3.8.1 Overview of SmartGrid-PTDRL

This research introduces a DRL-based PT framework designed to simulate and analyze cyber-physical attacks in smart grids, specifically replay attacks. The attack process is modeled as a Markov Decision Process (MDP) with three actions: *Stop*, *Record*, and *Replay*. The goal is to train an agent to learn the optimal timing and sequence of attacks to destabilize grid voltage under varying operational scenarios.

To achieve this, the researchers developed a cyber-physical co-simulation platform integrating physical, network, control, and attacker simulators.

This sandbox environment provides a robust experimental setup for training DRL agents. Experimental results demonstrated that the DRL agent can effectively find optimal attack paths, especially under high load demand, increased solar power generation, and significant weather variations, thereby threatening system stability.

3.8.2 Attack Operations

The SmartGrid-PTDRL framework simulates replay attacks using three main operations:

- **Record:** Captures the current system's PMU (Phasor Measurement Unit) data packets and stores them for subsequent replay.
- **Replay:** Launches replay operations when voltage approaches the CVR (Conservation Voltage Reduction) control limits (e.g., 0.95 or 1.05 pu), sending previously recorded PMU data to mislead the control center into issuing erroneous commands.
- **Stop:** Ceases any ongoing attack activities to maintain the current state.

Objective: Through carefully timed operations, the agent aims to disrupt voltage regulation and cause voltage levels to exceed safety ranges (typically 0.95–1.05 pu), resulting in grid instability.

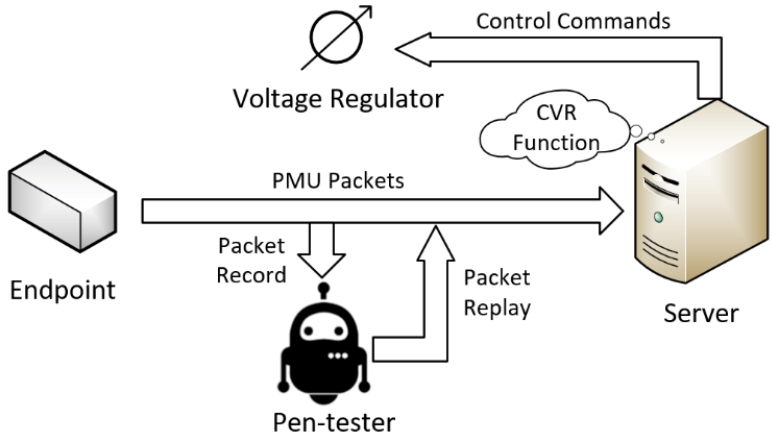


Figure 3.8: Replay attack scheme in the PT of smart distribution grids [7].

3.8.3 Framework Design

The SmartGrid-PTDRL framework is structured into three key components:

- **Environment:** Represents the physical components of the smart grid (e.g., transformers, loads), communication elements (e.g., sensors, communication interfaces), and the control center.
- **Feedback Mechanism:** The agent adjusts strategies based on feedback (e.g., voltage deviations) to maximize attack impact.
- **MDP Modeling:** The state space includes the grid's operational conditions. Observations replace full state access due to attack constraints. Actions include *Record*, *Replay*, and *Stop*, each with a fixed duration. Rewards are defined by the extent of voltage deviation using the System Average Voltage Magnitude Violation Index (SAVMVI).

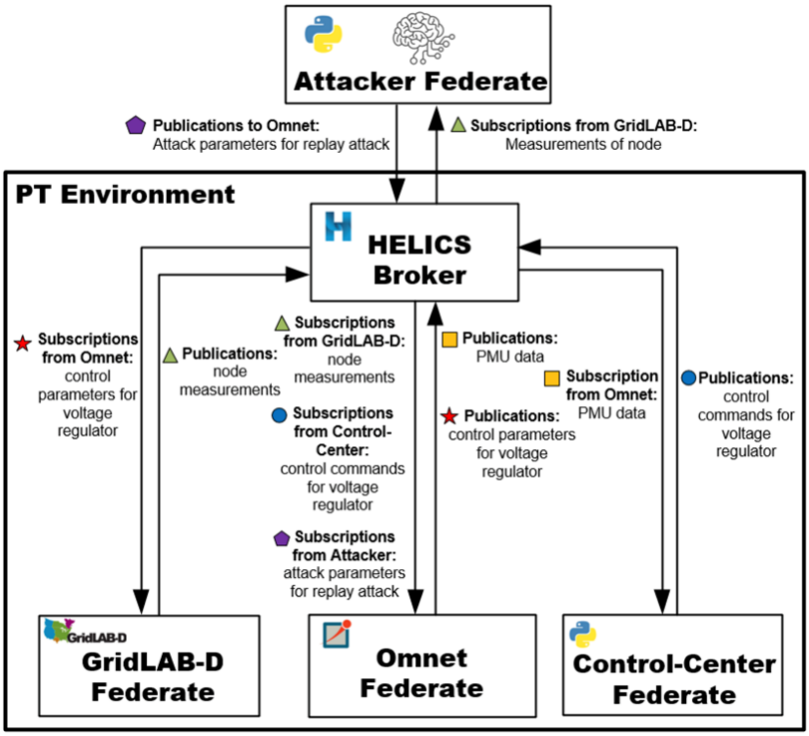


Figure 3.9: Cyber-physical testbed with smart distribution grids [7].

3.8.4 Experimental Validation and Findings

The researchers validated their approach using a co-simulation platform combining:

- GridLAB-D (power systems).
- OmNet++ (communication networks).
- Python-based control and attack modules.

These components were integrated using the HELICS framework for synchronization and data exchange.

Tests on an IEEE 13-bus system revealed how various factors, such as load levels, solar generation, and weather changes, affect PT difficulty. The DRL agent effectively exploited high-load scenarios, triggering voltage violations and system instability. However, lower load conditions exhibited limited attack impact.

3.8.5 Implications and Future Research

These findings demonstrate that DRL agents can autonomously learn effective attack strategies in smart grids, posing a significant threat to system stability under specific conditions. The research suggests that future work should:

- Investigate defense mechanisms against replay attacks.
- Improve the robustness of smart grid control systems.
- Explore advanced exploration strategies in DRL to enhance attack simulations.

3.9 ND3RQN

In 2022, Yue Zhang and colleagues published a paper titled *"Improved Deep Recurrent Q-Network of POMDPs for Automated Penetration Testing"* [8]. Notably, Zhou, the author who proposed NDSPI-DQN, is also one of the authors of this paper. Through modeling, algorithm improvements, and experimental validation, the authors introduced an enhanced reinforcement learning algorithm called ND3RQN, specifically designed for automated penetration testing in black-box scenarios.

3.9.1 POMDP-Based Modeling for Black-Box Penetration Testing

To better describe the complex environment of black-box penetration testing, the authors modeled penetration testing as a Partially Observable Markov Decision Process (POMDP). POMDP is more suitable for simulating penetration testing scenarios where testers have incomplete information about the target network. Based on this modeling, the authors proposed the improved Deep Recurrent Q-Network (ND3RQN), which enhances the traditional Deep Q-Network (DQN) with multiple improvements to boost its performance in partially observable environments.

3.9.2 Key Improvements in ND3RQN

The main improvements of ND3RQN include the following:

Incorporating the Long Short-Term Memory (LSTM) Structure

This structure enables the agent to remember historical information and make decisions based on past observations and actions. In a POMDP environment, the agent can only observe partial state information, leading to situations where the same observation might correspond to different global states. Historical information is thus critical for decision-making. LSTM captures long-term sequential dependencies, allowing ND3RQN to handle the uncertainty caused by partial observability more effectively.

Combining Double DQN and Dueling DQN Architectures

- **Double DQN:** Double DQN aims to reduce the overestimation problem of Q-values. Traditional DQN, which uses the same target network for both action selection and Q-value estimation, often overestimates Q-values. Double DQN separates the action selection process from the target Q-value generation process, significantly improving the accuracy of value estimation.

- **Dueling DQN:** Dueling DQN divides Q-value estimation into two components: a value function and an advantage function. This helps the agent better evaluate the importance of different states, thereby enhancing policy evaluation capabilities.

Introducing the Noisy Nets Exploration Mechanism

By adding random noise to the weights of neural networks, Noisy Nets introduces stochasticity into the agent's action selection process. Compared to the traditional ϵ-greedy strategy, this exploration mechanism is more effective in complex environments, helping the agent avoid getting trapped in local optima.

Additionally, ND3RQN retains the experience replay and target network mechanisms from the original DQN, which help break the temporal correlation between training samples and improve training stability.

3.9.3 Network Architecture of ND3RQN

As shown in Figure 3.10, the network architecture of ND3RQN cleverly integrates these improvements. The green sections represent the fully connected layers identical to the original DQN, the orange sections illustrate the LSTM structure, and the blue sections indicate the adversarial DQN structure with noise parameters. This architectural design enables ND3RQN to efficiently handle complex decision-making problems in partially observable environments.

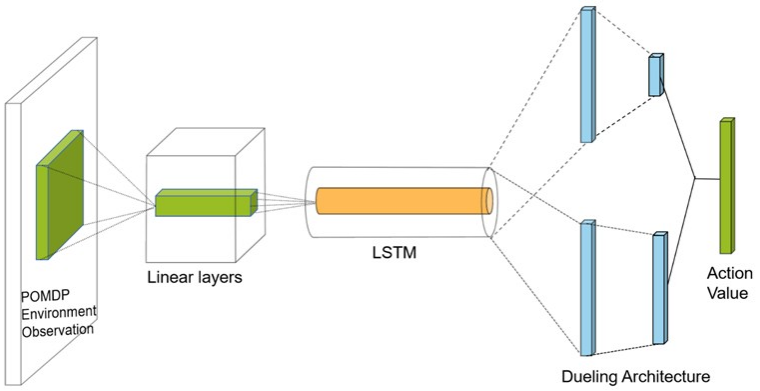


Figure 3.10: Network architecture of ND3RQN [8].

3.9.4 Experimental Evaluation

To validate the effectiveness of ND3RQN, the authors conducted extensive experiments using Microsoft's CyberBattleSim environment, an open-source cybersecurity simulator for automated penetration testing research.

Comparison with Other Algorithms

The performance of ND3RQN was compared with three other algorithms:

- Random action selection
- Standard DQN
- The latest improved DQN (NDSPI-DQN, proposed by Zhou Shicheng)

Performance in Different Network Environments

The experiments were conducted in network scenarios of different scales:

- **Small-scale networks:** Chain-10 and Chain-20.
- **Large-scale complex networks:** Randomly generated network topologies.

Experimental Results

- In small-scale network environments, ND3RQN demonstrated faster convergence and required fewer steps compared to other algorithms.
- In larger and more complex network environments, ND3RQN identified significantly more vulnerable nodes than its counterparts.

3.10 Improved-PenBox

In 2022, Alessandro Confido et al. published a paper titled *"Reinforcing Penetration Testing Using AI"* [32].

3.10.1 Overview

The research addresses the security challenges faced by the European Space Agency (ESA) in managing data systems during space missions. ESA developed an automated penetration testing tool called PenBox to simulate attacks and identify vulnerabilities in systems. PenBox incorporates a certain degree of automation by combining several existing penetration testing tools (e.g., Nmap, OpenVAS, and Metasploit) to automate tasks from network scanning to vulnerability discovery and exploitation. However, its automation is limited, as it relies on predefined attack scenarios and processes manually configured by security experts. During testing, PenBox sequentially

invokes penetration testing tools based on these predefined paths, resulting in a lack of flexibility to dynamically adjust strategies according to test results. This limitation makes it less effective in dealing with unknown network structures or new vulnerabilities. To overcome these constraints, the authors propose integrating reinforcement learning into PenBox, creating what they call "Improved-PenBox."

3.10.2 Implementation Details

The authors integrated the PenBox framework into the Network Attack Simulator (NAS) to leverage NAS's efficient simulation environment and reinforcement learning capabilities. Since the original PenBox functionality includes 52 tools and 363 modes, the authors selected the six most commonly used attack tools and modes (e.g., Nmap, Hydra, Metasploit) and mapped their operations to actions supported by NAS. By modifying NAS's network model and action interface, the authors ensured that it fully replicated the core functionalities of PenBox.

To achieve this, they defined Python configuration files describing network topology, host configurations, and tool operations. They also extracted corresponding attack cost and effectiveness data from the PenBox database and converted it into an input format that NAS could directly process. Using NAS's simulation environment, they employed its reinforcement learning interface to train a Deep Q-Network (DQN) algorithm and conducted experimental optimization of hyperparameters such as learning rate, discount factor, and exploration rate. The authors also made corresponding modifications to the reward function.

During the implementation, the authors created a network topology in NAS that closely resembled ESA's test environment. The network included multiple hosts, diverse service configurations, and vulnerability information to simulate typical attack scenarios. These scenarios covered intrusions from external networks to target hosts and internal attacks executed by PenBox within internal networks. Specifically, the simulated network comprised eight hosts, four services (SSH, FTP, HTTP, TCP), and a target machine with vulnerabilities. Through this design, the authors successfully replicated PenBox's functionality in NAS.

3.10.3 Experimental Results

By simulating real network environments in NAS, Improved-PenBox underwent multiple rounds of testing and progressively optimized attack paths. Starting from random initial attacks, the agent gradually learned to adjust

attack steps based on environmental feedback, ultimately generating an optimal penetration strategy. The experimental results demonstrated that by incorporating reinforcement learning, Improved-PenBox significantly reduced redundant operations and optimized the attack process.

3.11 OAKOC

In 2022, Rohit Gangupantulu et al. published a paper titled *"Using Cyber Terrain in Reinforcement Learning for Penetration Testing"* [33]. This research primarily explores how the concept of cyber terrain (such as firewalls) can be integrated into reinforcement learning (RL) methods to enhance the effectiveness of automated penetration testing. Existing research typically builds attack graphs based only on vulnerability information (e.g., CVSS scoring system), neglecting physical or logical obstacles in real networks (e.g., firewalls). Therefore, this paper proposes a method to incorporate cyber terrain into RL models, providing more realistic and effective solutions for penetration testing. We denote this OAKOC.

3.11.1 Methodology

The authors used Deep Q-Learning (DQN) as the core RL algorithm to simulate penetration testing via attack graphs. To incorporate cyber terrain into attack graphs, they proposed two key approaches:

- Reward adjustment.
- State adjustment.

Training was conducted using attack graphs generated by the MulVAL tool.

3.11.2 Attack Graph Construction

Attack graphs are graphical tools used in penetration testing to represent network structures. Each node in the graph represents a host or system state in the network, while each edge represents potential attack paths (state reachability). The authors used the MulVAL framework to automatically generate attack graphs. MulVAL is a logical reasoning engine that constructs attack graphs based on network configurations, identifying vulnerabilities and their impact on the network.

Using the OAKOC framework (Observation and Fields of Fire, Avenues of Approach, Key Terrain, Obstacles and Movement, Cover and Concealment), firewalls were modeled as obstacles. The authors simulated a network with:

- 122 hosts.
- An attack graph containing approximately 955 nodes and 2,350 edges.

This setup reflected a relatively complex network environment.

The CVSS vulnerability scoring system was used to derive state transition probabilities and rewards:

- State transition probability $P(s, a, s')$ was mapped from the CVSS "Attack Complexity" categories (Low, Medium, High) to 0.9, 0.6, and 0.3, respectively.
- Rewards were calculated based on vulnerability scores, with terminal state rewards set to 100, initial states at 0.01, and other states linearly scaled using depth-first search.
- If an action led to a state where the target node was unreachable, a negative reward (-1) was assigned.

3.11.3 Incorporating Cyber Terrain via Reward Adjustment

The study introduced cyber terrain into the reward system by adjusting reward values based on network obstacles (e.g., firewalls). By modifying the reward function $R(s, a)$, the agent received lower rewards when facing firewalls, simulating the effort required to bypass them. The formula is:

$$R(s, a) = R(s, a) + k(s)$$

where $k(s)$ is the reward adjustment value related to firewalls and depends on the communication protocol:

$$k(s) = \begin{cases} 0 & \text{No Firewall} \\ 0.8w & \text{FTP} \\ 0.6w & \text{SMTP} \\ 0.4w & \text{HTTP} \\ 0.2w & \text{SSH} \end{cases}$$

Here, w is a negative value used to adjust the reward reduction magnitude. This adjustment encourages the agent to avoid paths with firewalls or develop more sophisticated strategies to bypass them. The goal is for the agent to select more realistic attack paths during training rather than merely following the shortest path based on vulnerability scores.

3.11.4 Incorporating Cyber Terrain via State Adjustment

Another approach involved adjusting state transition probabilities to account for cyber terrain. In traditional Markov Decision Processes (MDPs), state transitions are deterministic. However, in this method, obstacles like firewalls significantly affect certain paths. This involved modifying state transition probabilities $P(s, a, s')$ to make paths traversing firewalls less likely, guiding the agent to find alternative routes. The formula is:

$$P(s, a, s') = P(s, a, s') \cdot k_1(s') \cdot k_2(s')$$

Factors affecting transition probability:

- $k_1(s')$ indicates whether a firewall is present:

$$k_1(s') = \begin{cases} 0.01 & \text{Firewall Exists} \\ 1.0 & \text{No Firewall} \end{cases}$$

- $k_2(s')$ represents the importance of the firewall's protocol:

$$k_2(s') = \begin{cases} 0.2 & \text{FTP} \\ 0.4 & \text{SMTP} \\ 0.6 & \text{HTTP} \\ 0.8 & \text{SSH} \\ 1.0 & \text{No Firewall} \end{cases}$$

This method further enhances the realism of the penetration testing model, forcing the agent to consider network obstacles in its decision-making process.

3.11.5 Experimental Results

The paper compared the performance of three different MDP models:

- **Baseline CVSS MDP:** No cyber terrain considerations.
- **Reward-adjusted MDP:** Cyber terrain was integrated through modified rewards.
- **State-adjusted MDP:** Cyber terrain was incorporated by adjusting state transition probabilities.

Results showed that incorporating cyber terrain increased the number of steps (hops) required for the agent to complete the penetration testing task, rather than simply following the shortest path. These additional steps reflected the agent's behavior of bypassing obstacles like firewalls.

The experiments also demonstrated the impact of the adjustment methods on total rewards:

- **Adding via Reward MDP:** The number of steps increased, but total rewards decreased due to reward attenuation.
- **Adding via State MDP:** Despite the increase in steps, the agent maintained higher total rewards, indicating its effective learning to navigate complex obstacles while still finding optimal paths.

Overall, the experimental results validate that by incorporating cyber terrain into attack graphs, the agent can more realistically simulate an attacker's decision-making process when facing obstacles in complex networks.

3.12 HDRL

In 2023, Qianyu Li and colleagues published a paper titled *"A Hierarchical Deep Reinforcement Learning Model with Expert Prior Knowledge for Intelligent Penetration Testing"* [34]. This paper proposed a Hierarchical Deep Reinforcement Learning (HDRL) model based on expert prior knowledge to address two critical challenges in intelligent penetration testing: the vast state and action spaces and inefficient exploration. This study uses a MDP model.

3.12.1 HDRL Model Design

The HDRL model addresses these challenges through two key strategies:

Hierarchical Structure

The HDRL model employs a hierarchical design approach, breaking down the complex task of penetration testing into distinct sub-tasks handled by specialized agents. These agents are dedicated to processing different types of penetration tasks, including local vulnerability exploitation, remote vulnerability exploitation, and connection operations. This design allows each agent to focus on a relatively smaller state and action space, thereby effectively reducing the computational complexity and learning burden of individual agents when dealing with large-scale network scenarios. This hierarchical

structure not only improves the task-handling efficiency of the model but also enhances its adaptability across various network environments.

Expert Prior Knowledge

To enhance the decision-making efficiency of agents and reduce the proportion of ineffective attempts during random exploration, the HDRL model incorporates an expert knowledge base built using rules and knowledge graphs. The expert knowledge base formalizes the experience of human penetration testing experts into actionable rules and retrievable knowledge nodes, providing agents with systematic action constraints and decision recommendations. Under the guidance of these rules, agents can swiftly eliminate invalid actions, while querying the knowledge graph enables them to access information on potential vulnerabilities in target hosts, allowing for the formulation of more precise strategies. This innovative integration of expert experience significantly boosts the model's learning efficiency and execution performance.

3.12.2 Key Features of the Model

- **Hierarchical Intelligent Penetration Testing Model:** The model categorizes penetration testing operations into three types, assigning them to distinct agents for processing. By leveraging deep learning techniques, the learning efficiency of the agents is effectively enhanced.
- **Integration of Expert Prior Knowledge:** The model transforms penetration testing experts' experience into two forms of prior knowledge: action constraints and action recommendations. This prevents agents from redundant sampling and aimless exploration in the environment.
- **Experimental Validation and Scalability:** Through simulations, the model demonstrated remarkable performance improvements in both small-scale and large-scale network scenarios, confirming its potential for practical applications.

3.12.3 Experimental Validation

In the experimental setup, researchers created a series of simulated network scenarios to evaluate the HDRL model's effectiveness and adaptability across networks of different scales. These scenarios included both small-scale networks (fewer than 10 nodes) and large-scale, complex networks (nearly 100

nodes). Experiments across various scenarios were designed to comprehensively assess the performance of the HDRL model, the impact of expert prior knowledge, and the scalability of the model in handling large-scale networks.

The results showed that the HDRL model significantly reduced the number of iterations required to achieve penetration testing objectives across all scenarios. Particularly, the model combining action constraints and expert recommendations (HDRL+AA+AC) performed exceptionally well in large-scale networks, achieving higher cumulative rewards in complex scenarios compared to traditional models.

3.13 DQfD-AIPT

In 2023, Wang Yongjie and his team published a paper titled *"DQfD-AIPT: An Intelligent Penetration Testing Framework Incorporating Expert Demonstration Data"* [9]. This paper proposed a framework named DQfD-AIPT.

3.13.1 Phases of the DQfD-AIPT Framework

The framework operates in three main phases:

Collection of Expert Knowledge

- Two methods are used to gather expert knowledge:
    1. The first method involves converting expert penetration testing experience into executable state-action pairs, which are stored in an expert knowledge base.
    2. The second method records expert operations across different network scenarios to provide diverse demonstration data, assisting the agent in understanding optimal penetration strategies under various circumstances.

Input and Utilization of Expert Data

- The collected expert data is fed into the system as demonstration data and serves as guidance during the initial training stage.
- The agent is pre-trained using this expert data, enabling it to quickly learn the decision-making process.

- The demonstration data is structured into tuples (state, action, reward, next state), consistent with the state transition mechanism in reinforcement learning, ensuring efficient knowledge acquisition in the early training stages.

Agent-Environment Interaction Training

- During the training phase, the agent interacts with a simulated network environment, performing penetration actions to alter network states and receiving reward feedback from the environment.
- The agent continuously optimizes its strategy based on its own experience and the expert demonstration data.

3.13.2 Key Improvements of the DQfD Algorithm

The DQfD-AIPT framework introduces several key improvements to enhance learning efficiency and stability:

Prioritized Experience Replay (PER)

- Important samples are given higher "priority", allowing the algorithm to replay more critical state transition data to enhance learning efficiency.
- The importance of experience data is calculated using temporal difference (TD) error, where higher TD errors indicate greater learning potential. This ensures that data contributing significantly to strategy improvement is prioritized.

N-Step Return Mechanism

- To better capture long-term reward information, the DQfD framework introduces an N-step return mechanism.
- N-step returns expand the range of reward propagation, allowing the influence of expert demonstration data to spread to earlier states. This helps the agent optimize strategies based on the long-term effects of a sequence of actions.

Supervised Loss

- During the pre-training stage, a supervised loss function is employed to guide the agent in imitating expert behavior and penalizing actions that deviate from expert decisions.
- This ensures the agent closely follows expert strategies during pre-training and further optimizes them during formal training.
- In the formal training phase, the supervised loss is removed, allowing the agent to explore strategies not covered by the expert demonstrations through reinforcement learning.

The training process of the DQfD algorithm is shown in Figure 3.11.

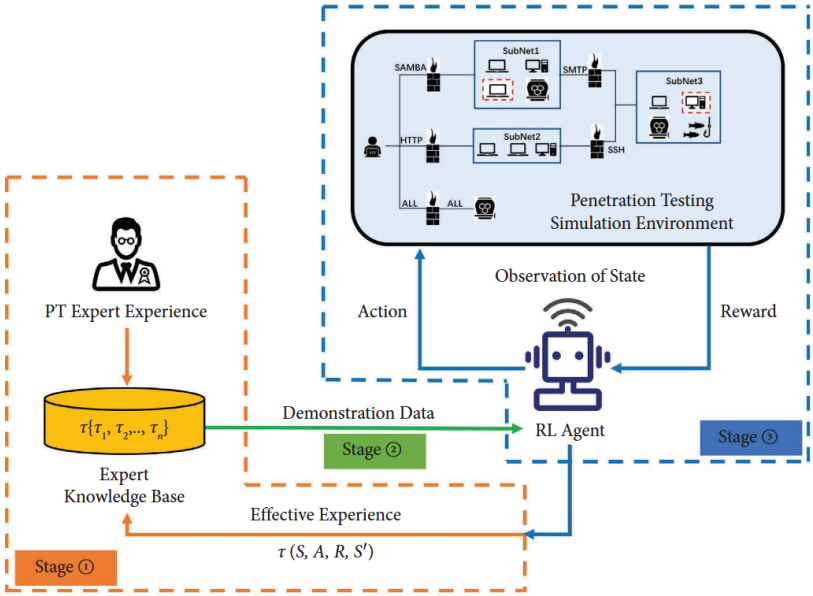


Figure 3.11: Framework structure of DQfD-AIPT [9].

3.13.3 Experimental Validation

The experiments utilized Microsoft's CyberBattleSim (CBS) platform, which simulates enterprise network environments. The network scenarios included:

- DMZ zone
- Trust-1 zone

- Trust-2 zone

Additionally, honeypot nodes were present in the environment. The objective of the penetration tests was to:

- Start from the DMZ zone.
- Avoid honeypot traps.
- Gain control of the database in the Trust-2 zone.

Honeypots are a network security defense mechanism designed to attract attackers and analyze their behavior. Performance evaluation metrics included:

- Average cumulative rewards.
- Probability of attacking honeypots.

3.13.4 Experimental Results

The results demonstrated significant advantages of the DQfD-AIPT framework over the standard DQN algorithm in several aspects:

Cumulative Rewards

- The DQfD algorithm converged faster and achieved higher cumulative rewards in each round of testing, indicating that the agent could efficiently identify the optimal attack path.

Honeypot Attack Probability

- Compared to the DQN algorithm, the DQfD framework significantly reduced the probability of attacking honeypots during training, showing that the agent could effectively recognize and avoid honeypots and other defense mechanisms.

Through these experiments, the researchers demonstrated that the integration of expert knowledge and reinforcement learning not only improved testing efficiency but also reduced errors, especially when dealing with complex network defenses like honeypots.

3.14 MDDQN

In 2023, Yi Junkai and Liu Xiaoyan published a paper titled *"Deep Reinforcement Learning for Intelligent Penetration Testing Path Design"* [35]. This research proposed a method called MDDQN (MulVAL Double Deep Q-Network), which aims to automate the design of penetration testing paths using deep reinforcement learning (DRL). The authors pointed out that Q-values in the DQN algorithm are prone to overestimation, which can lead to suboptimal policy updates and unstable behaviors. By improving the target function, DDQN mitigates this issue, making Q-values closer to their true values, which is why DDQN was employed. This research used a Markov Decision Process (MDP) model.

3.14.1 Working Mechanism of MDDQN

The MDDQN method is divided into three main phases:

Phase One: Attack Graph Generation

The authors used MulVAL to generate an attack graph for the target system. MulVAL constructs the attack graph based on the target system's host information, vulnerabilities, and their relationships, providing a comprehensive overview of all potential attack paths. Then, a Depth-First Search (DFS) algorithm is used to traverse the attack graph, identify all reachable paths, and organize this information into a transition matrix. The transition matrix details all reachable states, corresponding actions, and reward values for each action, laying the foundation for training the deep reinforcement learning model.

Phase Two: DDQN Agent Training

DDQN, an improved version of deep reinforcement learning, addresses the Q-value overestimation problem inherent in traditional deep Q-networks (DQN). After inputting the transition matrix generated in the first phase, the DDQN agent adopts an ε-greedy strategy for exploration and optimizes its learning process through experience replay. The goal is to maximize the cumulative long-term reward, thereby identifying the most effective attack path. By combining prior knowledge from the attack graph with a learning-based approach, MDDQN enhances the efficiency and stability of attack path planning.

Transition Matrix and Rewards

The research improved the transition matrix generation method proposed by Hu et al. [2]. It mapped all nodes in the attack graph to a matrix, including Common Vulnerability Scoring System Version 3 (CVSS 3) vulnerability values. It also included other operations, such as predefined scores for accessing files. All path information was retained, and a transition state matrix was generated by performing a DFS on the attack graph, which was directly fed into the model.

The transition matrix not only contained CVSS scores but also predefined operation scores and operation costs, making it more complex than Hu's research. The transition matrix provided positive rewards to the agent, guiding the training process and addressing the sparse reward problem.

3.14.2 Experimental Validation and Analysis

To validate the effectiveness of the MDDQN algorithm, the authors designed three network environments of different scales and conducted experiments. They compared the performance of MDDQN with other existing reinforcement learning algorithms, including:

- DQN
- DDQN
- DuelingDQN

The experimental scenarios involved multiple subnets, interconnected hosts, and various vulnerabilities to evaluate the performance of each algorithm in different settings.

Performance Comparison in Different Environments

The experimental results showed that MDDQN outperformed other algorithms in terms of convergence speed and stability, particularly in simpler test environments. MDDQN's advantage lies in its ability to leverage information from MulVAL attack graphs, allowing the agent to obtain more positive rewards in the early training stages, significantly accelerating convergence.

In more complex network environments, MDDQN still exhibited higher stability and scalability. While other algorithms experienced significant fluctuations during training due to the complexity of the environment, MDDQN effectively reduced such instability by guiding exploration through attack graphs and efficiently identifying the optimal attack paths.

Addressing the Sparse Reward Problem

Additionally, MDDQN partially addressed the sparse reward problem commonly found in deep reinforcement learning. In traditional penetration testing scenarios, positive rewards are usually granted only when a critical target is successfully breached, making it challenging for the agent to gain sufficient rewards during learning.

By incorporating prior knowledge from attack graphs, MDDQN enabled the agent to receive more frequent positive rewards during exploration, accelerating the overall training speed and effectiveness.

3.15 INNES

In 2023, Qianyu Li and colleagues published a paper titled *"INNES: An Intelligent Network Penetration Testing Model Based on Deep Reinforcement Learning"* [36], aiming to propose an intelligent penetration testing model called INNES.

3.15.1 Overview of INNES

The INNES model describes the penetration testing process based on the Markov Decision Process (MDP). Unlike traditional methods, the INNES model eliminates the reliance on prior knowledge of the target network structure. Instead, it gradually builds the network's state representation during the exploration process, enhancing the model's adaptability and portability.

To reduce ineffective exploration, the authors proposed the DQN_valid algorithm, which restricts the action space of the agent to avoid meaningless exploration during testing, thereby improving learning efficiency and decision accuracy.

3.15.2 DQN_valid: Optimized Action Space

The core idea of DQN_valid is to limit the agent's action choices to a "valid action space" at each decision step, rather than the entire action space.

Definition of Valid Action Space

As the testing progresses, the authors define:

- A set of known nodes K.
- A dynamically expanding valid action set V.

The valid action space progressively includes more actionable steps, ensuring the agent does not waste effort on redundant or meaningless actions.

Reward Function Optimization

The reward function in DQN_valid is designed as follows:

- Successful actions are rewarded positively.
- Failed actions are penalized to discourage ineffective exploration.

3.15.3 Experimental Evaluation

To validate the effectiveness of the INNES model, the paper conducted experimental evaluations in various network scenarios, including:

- Real-world network environments.
- Open-source simulation environments such as Microsoft's CyberBattleSim platform.

INNES demonstrated outstanding performance in these experimental scenarios. Moreover, the DQN_valid algorithm significantly reduced the number of ineffective explorations by the agent, accelerated the training process, and enhanced overall performance, making INNES more efficient and reliable for penetration testing tasks.

3.15.4 Portability Across Network Environments

A key result of the experiments is the validation of the INNES model's portability across different network environments. The model could efficiently adapt to unseen scenarios.

For instance, in the **Network1** scenario, the transferred model completed the task in just 11 iterations.

Portability is crucial for automated penetration testing, as network environments are often dynamic. Models capable of adapting to new environments without retraining can significantly reduce deployment time and costs.

3.16 HER-PT

In 2024, Mingda Li and others published a paper titled *"HER-PT: An Intelligent Penetration Testing Framework with Hindsight Experience Replay"* [10], which detailed an innovative penetration testing framework called HER-PT.

The HER-PT framework combines the mechanism called **Hindsight Experience Replay (HER)** with deep reinforcement learning models. By reconstructing and transforming failed experiences into positively rewarded ones, the model learns from failures, thereby significantly improving learning efficiency and success rates. This approach addresses the issue of slow training and suboptimal performance caused by a lack of successful penetration testing experiences. HER introduces a "goal resetting" mechanism, enabling unsuccessful attempts to be redefined as successes during training. Specifically, it selects a state s_{t+1} from a failed experience as a new goal g', updates the reward $r_{g'}$ based on g', and stores the reconstructed experience as a new sample. Both the original failed experience and the reconstructed samples are added to the experience replay buffer, increasing the proportion of positive samples. This method effectively tackles the sparse reward problem by enhancing the ratio of positively rewarded samples in the replay buffer.

3.16.1 Details of HER-PT

As shown in Figure 3.12, The HER-PT framework consists of three core modules: **penetration testing scenario construction, optimal strategy generation, and practical attack application**.

Penetration testing scenario construction

In this module, HER-PT creates a multi-host network environment that simulates complex real-world scenarios, including subnets, host configurations, and firewall rules. By emulating different network topologies and device configurations, the framework can construct diverse and complex cybersecurity environments to evaluate the model's performance under various conditions.

Optimal strategy generation

In this module, HER-PT integrates HER technology with the **Deep Q-Network (DQN)** algorithm to improve sample utilization by reconstructing experience data. This allows the agent to learn optimal attack strategies more efficiently. Specifically, HER converts failed experiences into useful learning samples, increasing the proportion of successful experiences and accelerating the learning process.

Practical attack application

In this module, HER-PT integrates the **Metasploit framework**, translating abstract penetration strategies into specific attack commands. This enables the execution of penetration tests in real network environments, verifying the model's effectiveness.

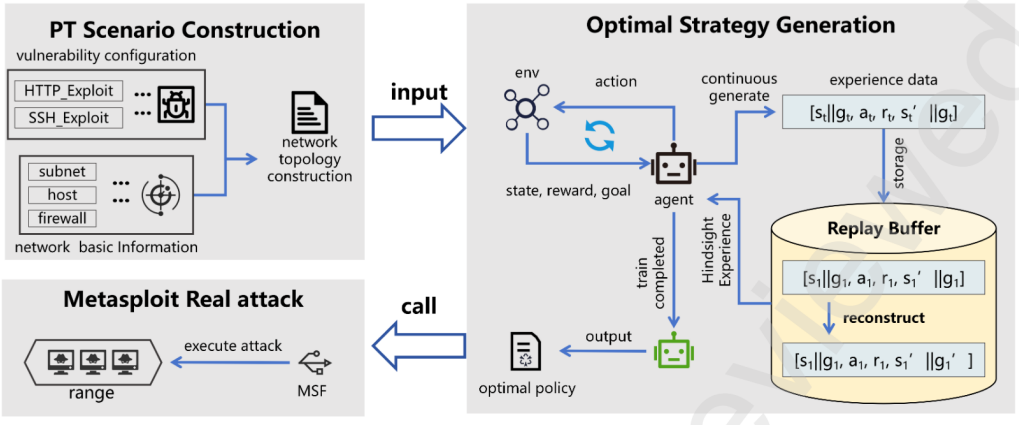


Figure 3.12: Framework structure of HER-PT [10].

3.16.2 Experimental Evaluation

The experimental methods and processes outlined in the paper focus on validating the adaptability, training efficiency, and practicality of the HER-PT framework in different network environments. The experiments are designed around three categories, conducted using the NAS simulator and the Metasploit platform:

- **Adaptability experiments:** Dynamic network scenarios with different frequencies of changes (low, medium, high) are constructed using the NAS network attack simulator. The performance of HER-PT is compared with the traditional rule-based approach MulVAL in terms of convergence speed and success rates, evaluating HER-PT's adaptability in dynamic network environments.
- **Efficiency experiments:** Using the NAS simulator, HER-PT is compared with classical DQN and its variants (e.g., Double DQN and Dueling DQN) under sparse reward environments. Different reward mechanisms, network structures, and experience replay strategies are analyzed to systematically examine the effect of HER in accelerating model convergence and enhancing efficiency.

- **Practicality experiments:** Abstract penetration strategies generated by HER-PT on the NAS platform are translated into executable attack commands on the Metasploit platform. Penetration testing tasks are performed in six real-world network scenarios of varying scales and complexity. The experiments evaluate HER-PT's penetration success rates and operational efficiency, providing a comprehensive assessment of its applicability and reliability in real-world penetration testing tasks.

3.17 DynPen

In 2024, Qianyu Li et al. published the paper *"DynPen: Automated Penetration Testing in Dynamic Network Scenarios using Deep Reinforcement Learning"* [19]. Most of the existing research in the field of automated penetration testing focuses on penetration testing in static network environments, but real-world network environments are dynamic, making it difficult for existing methods to adapt to this complexity. This research study proposes an automated penetration testing framework called DynPen, designed to meet the challenges of dynamic environments.

3.17.1 Research Methods and Implementation Details

DynPen is based on DRL and designed specifically for dynamic network environments. The overall structure of DynPen includes four modules:

- Penetration Testing Agent (PT Agent)
- Trajectory Logging Module
- Monitoring Module
- Backtracking Module

These modules work together to enhance the system's adaptability to dynamic environments.

Penetration Testing Agent (PT Agent)

The PT agent of DynPen uses DRL to make decisions, aiming to learn optimal penetration paths through continuous interaction with the environment. The following features are incorporated:

- **HDRL Structure:** The PT agent is designed with a two-layer Hierarchical Deep Reinforcement Learning (HDRL) structure. The upper layer selects suitable subtasks (e.g., vulnerability exploitation, and credential connections), while the lower layer consists of three sub-agents for local vulnerability exploitation, remote vulnerability exploitation, and credential connections. This hierarchical design reduces the action space, enhancing learning efficiency and testing performance.
- **Expert Knowledge Base Assistance:** To improve sample efficiency, DynPen incorporates an expert knowledge base to assist and constrain penetration testing strategies during the reinforcement learning process. This helps the agent obtain effective data faster by reducing unnecessary exploration.

Trajectory Logging Module

The trajectory recording module first logs all decisions made by the agent, including state, action, and timestamp information. Based on the trajectory data, critical node analysis can then be conducted. This involves identifying the nodes in the network topology that have the greatest impact on penetration paths, such as highly connected nodes or nodes with high betweenness centrality. By doing so, the path selection in the backtracking module can be optimized, enabling the identification of nodes that need to be re-explored based on the trajectory data.

Monitoring Module

The monitoring module detects changes in the network, focusing on the status of critical nodes controlled by the agent. Key features include:

- **Change Detection:** Periodic scanning of nodes and comparison with historical data to detect changes.
- **Critical Node Identification:** Determines critical nodes (e.g., nodes with high connections or connected to high-value nodes) and prioritizes their monitoring to reduce resource usage while ensuring sensitivity to important changes.

Backtracking Module

The backtracking module addresses unexpected changes in dynamic network environments by:

- **Resuming Penetration Operations:** Identifies affected nodes and resumes penetration testing from those points.
- **Alternate Path Exploration:** Allows the agent to attempt other paths if penetration fails, ensuring continuity of testing tasks.

3.17.2 Experimental Design and Results Analysis

The author, building upon the static scenarios from previous research, namely HDRL+HF, designed four types of simulated dynamic networks to validate the effectiveness of DynPen through simulation experiments. These include changes in node attributes, changes in network connectivity, variations in the number of nodes, and the introduction of honeypot nodes. The main experimental designs and results are as follows:

Comparison Methods

The following models were compared:

1. **Random Method:** A strategy that selects penetration paths randomly.
2. **DQN Valid Model:** A simplified model using Deep Q-Networks. (from Qianyu Li's research on 2023, INNES [36])
3. **HDRL+HF Model:** A hierarchical deep reinforcement learning model used as a benchmark. (from Qianyu Li's research on 2023, HDRL [34] and INNES [36])
4. **DynPen Model:** The proposed dynamic adaptive penetration testing system.

Experimental Results

The experiments revealed that, although the existing HDRL+HF model performed well in static scenarios, its lack of environment monitoring and adjustment mechanisms resulted in significantly poorer performance in dynamic scenarios compared to DynPen. On the other hand, the random strategy without a learning mechanism performed the worst across all scenarios.

3.18 DRLRM-PT

In 2024, Yuanliang Li et al. published a paper titled *"Knowledge-Informed Auto-Penetration Testing Based on Reinforcement Learning with Reward Machine"* [11]. The paper proposed the **DRLRM-PT** framework, which integrates Reward Machines (RMs) with Deep Reinforcement Learning (DRL) to address several critical issues in automated penetration testing, including:

- **Low sampling efficiency**: Existing methods require extensive environmental interactions to derive optimal strategies.
- **Complexity of reward functions**: Defining clear and effective reward mechanisms for penetration testing is challenging.
- **Lack of interpretability**: Current RL models struggle to provide transparent and interpretable decision rationales.

3.18.1 Overview of DRLRM-PT Framework

The DRLRM-PT framework leverages reward machines as a driving mechanism. Its main methodologies include:

- **Task decomposition**: Reward machines are used to break down complex penetration testing tasks into multiple subtasks, each with an independently defined reward function.
- **Knowledge embedding**: Information from cybersecurity knowledge bases such as MITRE ATT&CK is embedded to provide explicit guidance.

This approach improves sampling efficiency during training, enhances the flexibility of reward mechanism design, and increases the interpretability of the generated strategies.

3.18.2 Reward Machines (RMs)

Reward Machines are one of the key innovations in this research. Acting as state machines, they encode domain knowledge by breaking down complex penetration testing tasks into manageable stages and assigning rewards for actions within these stages. With RMs, a complex penetration testing task can be expressed as a series of interrelated subtasks with specific goals. This task decomposition enables the agent to efficiently learn optimal actions during training, significantly reducing ineffective exploration.

Types of Reward Machines

The paper introduces two distinct RM designs, R1 and R2, to guide the penetration testing process at different levels of granularity:

- **R1**: A simpler design that divides penetration testing tasks into three stages: discovering new credentials, connecting to new nodes, and escalating node privileges to control more assets.
- **R2**: A more detailed design that adds an extra subtask—discovering new nodes before searching for credentials. The finer task decomposition in R2 provides clearer guidance for agents, thereby improving strategy training efficiency.

3.18.3 POMDP Modeling and DQRM

Specifically, the DRLRM-PT framework models the penetration testing task as a Partially Observable Markov Decision Process (POMDP) and uses deep reinforcement learning to find the optimal strategy. Throughout the process, reward machines are employed to decompose tasks, generate subtask sequences, and assign independent reward functions to each subtask, improving the agent's learning efficiency.

Introduction of DQRM

In implementation, the research team proposed an enhanced version of the traditional Deep Q-Network (DQN), named **DQRM** (Deep Q-learning with Reward Machine). DQRM effectively incorporates domain knowledge through reward machines, enhancing the DQN.

Roles of Reward Machines in DQRM

RMs play two primary roles in this framework:

- Guiding the agent through the complex path from goal-setting to goal achievement.
- Providing specific and independent reward mechanisms for each intermediate step.

This design enables DQRM to exhibit significant efficiency advantages in handling complex tasks.

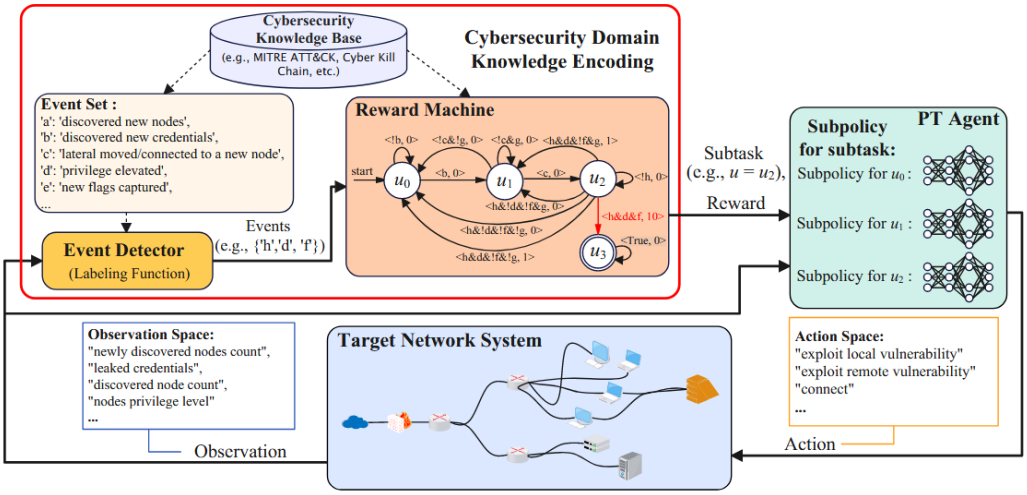


Figure 3.13: Architecture of the knowledge-informed AutoPT framework (DRLRM-PT) [11].

3.18.4 Experimental Evaluation

To validate the effectiveness of the proposed method, the research team conducted experiments using Microsoft's open-source network simulation platform, **CyberBattleSim**.

Tested Network Environments

The experiments involved two typical network environments:

- **CyberBattleChain**: A sequential network simulating scenarios where the agent needs to gradually escalate node privileges.
- **CyberBattleToyCtf**: A more complex mesh network with multiple nodes and sensitive data, simulating more realistic enterprise network environments.

These experiments aimed to evaluate the performance of DRLRM-PT with integrated reward machines in different network types and compare it to traditional DQN models.

3.18.5 Experimental Results

The results showed that the DQRM model enhanced with reward machines significantly outperformed baseline DQN models in terms of training efficiency and penetration testing performance.

Performance Comparison

During training, agents using reward machines exhibited the following advantages:

- Completed complex tasks with fewer steps.
- Achieved substantially higher cumulative rewards compared to traditional models.

Impact of Reward Machine Designs

Furthermore, the experiments revealed that more detailed reward machine designs (such as R2) further improved agent performance, as more precise task decomposition provided clearer guidance, enabling the agent to identify the optimal attack path more quickly.

3.19 DL-IAPTS

In 2024, Abdul Samad and others published a paper titled *"Advancements in Automated Penetration Testing for IoT Security by Leveraging Reinforcement Learning"* [37]. This paper proposed an Intelligent Automated Penetration Testing System (IAPTS) based on deep reinforcement learning to address the inefficiencies and long duration of automated penetration testing in medium-to-large-scale networks. We denote this DL-IAPTS.

3.19.1 Methodology

The authors modeled the penetration testing environment as a Partially Observable Markov Decision Process (POMDP), which is well-suited for penetration testing scenarios. The optimization of attack paths was carried out using Deep Q-Networks (DQN). By combining the advantages of Q-learning and neural networks, DQN can continuously learn from the network environment to identify optimal attack paths.

To ensure the system can adapt in real-time to changes in network states, the study introduced a dynamic threat matrix generated using tools like *MulVAL* and *Shodan* (referred to as "Shadov" in the original text but later inferred to mean Shodan). This matrix continually updates network node information to support the training of the reinforcement learning model. DL-IAPTS employs the Common Vulnerability Scoring System (CVSS) to assign reward values to each attack node. CVSS scores quantify the severity

of vulnerabilities and convert them into rewards, motivating the system to prioritize more critical attack paths.

The specific modules of DL-IAPTS include:

1. **Dynamic Network Intrusion Matrix Generation:** The dynamic matrix represents the current state of network intrusions. The study used MulVAL and Shodan tools to construct a continuously updated threat matrix for reinforcement learning training and updates.

2. **Deep Q-Learning Network Training:** The system utilizes DQN to train on the dynamic matrix and automatically generate optimal attack paths. DQN evaluates the rewards of different states and actions, continuously adjusting strategies to ensure the selected path maximizes the overall attack efficiency.

3. **POMDP Model Construction:** The POMDP model describes state transitions and partial observability in penetration testing. In this model, the system learns penetration testing strategies through stochastic state transitions, helping it make optimal decisions in complex environments.

3.19.2 Experiments

To validate the effectiveness of DL-IAPTS, the researchers conducted multiple experiments. In these experiments, they used the Shodan tool to collect potential vulnerabilities, open ports, product information, and protocol data from the network. They then employed the MulVAL tool to generate attack graphs for the network environment, which were used to train and optimize the deep reinforcement learning model.

The primary goal of the experiments was to optimize attack paths through the reinforcement learning model, thereby improving the efficiency and quality of penetration testing. Researchers used the MulVAL system to generate attack graphs and DQN to optimize them. In the experiments, DL-IAPTS successfully generated attack paths automatically and identified high-priority attack nodes. The tests covered various types of network nodes, including file servers, web servers, and workstations. The attack path for each node was optimized by DQN, and the system leveraged a reward mechanism to prioritize and attack high-risk nodes.

3.20 Overall Analysis of DQN Approaches

Based on the summary of previous papers, we can identify that the main challenges in the research of automated penetration testing focus on two major issues:

1. The application of large and complex network environments.
2. The sparse reward problem.

When traditional DQN is applied to automated penetration testing, it inherently encounters these two problems. Therefore, most researchers have proposed solutions specifically targeting these challenges.

We can also observe that some studies are from the same author or research team. For example:

- **NDSPI-DQN** (2021) [4] and **ND3RQN** (2022) [8] were developed by the same team. Along with **DQfD-AIPT** (2022) [9], all three studies originate from the same research institution.
- **INNES** (2023) [36], **HDRL** (2023) [34] and **DynPen** (2024) [19] are from the same author.
- **CJA-RL** (2021) [5] and **OAKOC** (2022) [33] are also from the same author.

We will compare these studies collectively. It is evident that research from the same team or institution often shares certain methodologies or technologies to some extent.

3.20.1 Analysis of NDSPI-DQN, ND3RQN and DQfD-AIPT

Let us first look at **NDSPI-DQN** [4] and **ND3RQN** [8]. As shown in Table 3.1, the technical features and differences between these two studies are listed. It can be seen that ND3RQN was developed based on the results of NDSPI-DQN. Compared to NDSPI-DQN, ND3RQN demonstrates greater practicality.

About **DQfD-AIPT** [9], when compared to NDSPI-DQN [4] and ND3RQN [8], it uses *expert knowledge* as the driving force, which is also a commonly used approach. Later on, we will provide a detailed comparison of all studies that apply similar techniques. This method abstracts expert knowledge

Table 3.1: Comparison of NDSPI-DQN and ND3RQN

Technical Method	**NDSPI-DQN (2021)** [4]	**ND3RQN (2022)** [8]
Noise Network	Uses Gaussian noise for more structured exploration	Uses random noise for stronger exploration
Experience Replay	Prioritized experience replay based on TD time error, prioritizing important samples	Random sampling to avoid data correlation; uses long short-term memory (LSTM) to process historical information
Adversarial Architecture	Uses an adversarial network to decompose Q-value into state value (Value) and advantage (Advantage) to improve policy evaluation capabilities	Similarly introduces adversarial architecture but adds optimization for state observation based on POMDP requirements
Model	MDP	POMDP
Reward Mechanism	Combines internal and external rewards to design cumulative targets under sparse rewards	Adjusts the Q-value computation formula to resolve overestimation issues and enhances reward feedback
Action Space Optimization	Decouples action vectors into target hosts and specific operations to reduce the complexity of combined actions	Introduces historical memory and uses LSTM to optimize decision-making under the same observation but different histories

into standard quadruples, then employs TD-error-based prioritized replay, combined with n-step returns and supervised loss, to ensure the training effectiveness of the agent. However, I believe this research heavily relies on expert knowledge, requiring a substantial amount of it to achieve good results, which makes it less practical.

3.20.2 Analysis of INNES, HDRL and DynPen

INNES and HDRL can be considered as the precursor studies of DynPen. Each of these three studies has its own strengths and focuses. The results of the comparative analysis are shown in the Table 3.2.

Table 3.2: Comparison of INNES, HDRL, and DynPen

Feature	**INNES (2023)** [36]	**HDRL (2023)** [34]	**DynPen (2024)** [19]
Core Idea	Focusing on DQN_valid to constrain invalid actions.	Hierarchical DRL (HDRL) model incorporating expert knowledge graphs and rules to optimize learning efficiency.	A model for dynamic network environments with trajectory recording, real-time monitoring, and backtracking modules for adaptive learning.
Challenges Addressed	Unreasonable PT process characterization and low data efficiency of DRL.	Large state/action spaces causing difficulty in strategy learning and low random exploration efficiency.	Slow convergence and inability to adapt to dynamic network environments.
State Representation (MDP/POMDP)	Based on Markov Decision Process (MDP), assuming fully observable states.	MDP-based, using expert knowledge to decompose tasks into smaller state spaces for stability.	MDP-based with dynamic monitoring to update state representation.
Action Space Characteristics	Action space dynamically constrained by DQN_valid to avoid invalid exploration.	Actions divided into three categories (local exploitation, remote exploitation, connection) handled by multi-layer agents.	Includes complex actions (e.g., scanning, exploitation, connection) with dynamic adjustments of valid action sets.
Reward Mechanism	Immediate rewards based on the effectiveness of current actions.	Immediate rewards + advice-driven rewards based on expert optimization.	Combines immediate rewards with trajectory-level evaluation for overall task optimization.

Feature	**INNES (2023)** [36]	**HDRL (2023)** [34]	**DynPen (2024)** [19]
Dynamic Adaptability	Limited adaptability, suitable for static or simple network scenarios.	Weak adaptability to dynamics, primarily for static large-scale networks.	High adaptability with trajectory recording and backtracking modules for dynamic environments.
Integration of Expert Knowledge	No expert knowledge, purely relies on DRL.	Expert knowledge graphs and rule bases guide action selection, reducing blind exploration.	Combines expert-defined trajectory rewards and dynamic modules with historical data for strategy optimization.
State/Action Space Scale	Simple, constrained by DQN_valid to limit state and action size.	Hierarchical structure decomposes large state and action spaces into smaller subtasks.	Suitable for large dynamic network environments, with state and action spaces dynamically adjusted.
Experimental Environment	Simulated static network environments for automated penetration testing experiments.	Experiments conducted in large-scale static network simulation environments.	Dynamic network simulation environments, emphasizing real-time adaptability and fast strategy generation.
Model Complexity	Low, mainly dependent on single agent and simple action constraints.	Medium, employs multi-agent structure and expert knowledge integration.	High, combining dynamic monitoring, backtracking modules, and complex trajectory rewards.
Applicable Scenarios	Static small-scale network scenarios, suitable for well-defined testing tasks.	Static large-scale networks, suitable for task decomposition and optimization in complex topologies.	Dynamic network scenarios, ideal for rapidly changing real-world environments.

Feature	**INNES (2023)** [36]	**HDRL (2023)** [34]	**DynPen (2024)** [19]
Advantages	Easy to implement and deploy; highly portable.	Suitable for large-scale scenarios, significantly improves learning efficiency; reduces computational resources.	Highly dynamic adaptability, fast convergence; efficient strategy generation in complex scenarios.
Limitations	Not suitable for dynamic scenarios; weak support for large-scale environments.	Strongly dependent on expert knowledge, which may limit model autonomy; weak dynamic adaptability.	High complexity may increase implementation and deployment costs; relies on extensive simulation setups and historical data accuracy.

3.20.3 Analysis of Research Related to MulVAL

From the previous review, we can also see that using MulVAL attack graphs is a relatively common approach. Studies utilizing this method include Autopentest-DRL (2020) [2], CJA-RL (2021) [5], MDDQN (2023) [35], ASAP (2020) [3], DL-IAPTS (2024) [37], and OAKOC (2022) [33]. A detailed analysis and comparison are shown in Table 3.3.

Table 3.3: Comparison of frameworks related to MulVAL

Framework	**Features**	**Application Scenario**	**Model**	**Experiment**
Autopentest-DRL (Hu et al., 2020) [2]	Based on MulVAL and DQN, uses Shodan to generate network topology, optimizes penetration paths using deep reinforcement learning	Small-scale networks	MDP	Tested in real network scenarios, path discovery accuracy of 86%
CJA-RL (Gangupantulu et al., 2021) [5]	Introduces "Crown Jewel Analysis (CJA)" and network structure analysis, integrates MITRE framework	Enterprise network key asset protection analysis	MDP	Multi-path optimization experiments showing key nodes and bottlenecks
MDDQN (Yi & Liu, 2023) [35]	Improved DQN algorithm with Double Deep Q-Network (DDQN), solves convergence issues and enhances efficiency	Large-scale network penetration testing path design	MDP	Tested in networks of various scales, faster and more stable convergence
ASAP (Chowdhary et al., 2020) [3]	Automated penetration testing framework based on DQN, utilizes CVSS scoring to model rewards and transition matrix	Security vulnerability identification and path planning for large enterprise networks	MDP	Tested on 300 hosts, generates paths in under 70 seconds
DL-IAPTS (Samad et al., 2024) [37]	Task modeled using POMDP, emphasizes learning reuse to improve efficiency of penetration testing strategy generation	Security auditing and penetration testing of small to medium-sized networks	POMDP	Focuses on small to medium-sized networks, highlights learning reuse effectiveness
OAKOC (Gangupantulu et al., 2022) [33]	Combines OAKOC terrain analysis framework, models terrain characteristics into attack graphs and reward mechanisms	Penetration path analysis for large, complex networks, emphasizing the impact of network terrain	MDP	Tested on networks with over 1000 nodes, includes terrain analysis

In summary, it is evident that using MulVAL attack graphs is one of the key approaches in the field of automated penetration testing. From the table, we can also see that applying these methods to larger-scale networks and utilizing POMDP models are emerging trends for future research. The design of reward mechanisms is becoming increasingly diverse and sophisticated. These mechanisms significantly enhance the practicality and relevance of penetration testing strategies.

3.20.4 Analysis of Research Related to HER, PER and Expert

Next, we analyze studies that have utilized experience-related techniques. From the previous review, we can observe the following: HER-PT (HER) [10], NDSPI-DQN (PER) [4], DQfD-AIPT (PER) [9], DynPen (Expert assistance) [19], MDDQN (Expert assistance) [35], and ND3RQN (LSTM) [8]. All these studies employ methods associated with experience. We conducted a comparative analysis of these studies, as shown in Table 3.4.

Table 3.4: Comparison of frameworks and methods related to HER, PER and Expert

Framework or Method	**Core Features**	**Application Scenarios and Advantages**	**Model Type and Task Definition**	**Improved Replay Strategies**	**Experiment Environment and Tools**	**Practicality and Application Scenarios**
HER-PT [10]	Uses Hindsight Experience Replay (HER) to modify the experience pool, reconstructs failed sample goals, increases the proportion of positive samples in sparse reward scenarios, accelerates model convergence.	Suitable for multi-host networks, optimizing tasks in sparse reward environments for automatic penetration testing, accelerating the learning of optimal attack paths. Mainly addresses sparse reward and multi-host adaptation problems.	MDP, task defined as a sparse reward path planning problem.	Goal redefinition based on HER, reconstructing failed sample goals, enhancing sample utilization, accelerating convergence in sparse reward environments.	Uses the NAS simulator for multi-host scenarios and optimizes Metasploit API for actual attacks.	Suitable for multi-host sparse reward scenarios, emphasizing the integration of automated strategies with practical deployments.

Framework or Method	Core Features	Application Scenarios and Advantages	Model Type and Task Definition	Improved Replay Strategies	Experiment Environment and Tools	Practicality and Application Scenarios
NDSPI-DQN [4]	Combines PER (Prioritized Experience Replay) with TD-error and various DQN extensions (e.g., NoisyNet, Dueling Architecture) to mitigate issues with large action spaces and sparse rewards.	Aimed at large-scale networks, addressing low exploration efficiency caused by large action spaces, optimizing algorithm convergence speed and scalability.	MDP, task defined as optimization of penetration strategies in large-scale networks.	Uses PER to prioritize samples, combined with NoisyNet to increase sampling probability of critical samples.	Validates algorithm convergence and scalability in large-scale network scenarios.	Suitable for complex networks, particularly performing well in large action spaces.
DQfD-AIPT [9]	Integrates expert experience, balances expert and interactive data, and incorporates PER.	Optimizes path planning and avoids honeypots in complex scenarios, enhancing training efficiency and robustness while preventing expert overfitting.	MDP, task defined as path planning to avoid honeypots in complex networks.	Balances expert and interactive samples in PER to optimize sample diversity and address overfitting issues.	Uses CyberBattleSim to simulate complex networks with honeypots.	Targets complex environments, emphasizing the impact of expert samples on honeypot avoidance and path optimization.

Framework or Method	Core Features	Application Scenarios and Advantages	Model Type and Task Definition	Improved Replay Strategies	Experiment Environment and Tools	Practicality and Application Scenarios
ND3RQN [8]	Incorporates LSTM into deep reinforcement learning, enhancing memory of state history, suitable for partially observable Markov decision processes (POMDPs).	Black-box penetration testing scenarios, optimizing strategy formulation in environments with scarce observational information and uncertainty, improving generality and adaptability.	POMDP, task defined as strategy optimization in black-box penetration testing.	Replay buffer with LSTM records historical trajectory information, suitable for partially observable environments.	Uses CyberBattleSim to validate strategy generality in black-box scenarios.	Suitable for information-scarce and dynamic environments, offering excellent adaptability and generality.

Framework or Method	Core Features	Application Scenarios and Advantages	Model Type and Task Definition	Improved Replay Strategies	Experiment Environment and Tools	Practicality and Application Scenarios
DynPen [19]	Dynamically adapts to networks by adjusting strategies based on historical decision trajectories and an environment monitoring module, dynamically adjusting action spaces in response to environmental changes.	Adapts to dynamic network changes in penetration testing, significantly improving adaptability and learning efficiency in rapidly changing scenarios.	MDP, task defined as path planning and decision-making in dynamic network environments.	Incorporates a dynamic adjustment module, leveraging replayed historical trajectories to adjust decision spaces.	Validates decision adaptability in dynamic network simulation environments.	Emphasizes quick strategy adjustment in dynamic networks, suitable for real-world environments.
MDDQN [35]	Combines attack graphs and Double Deep Q-Networks (DDQN) to provide prior rewards, partially addressing sparse reward issues.	Suitable for attack path design, leveraging attack graphs to optimize reward mechanisms and accelerate convergence stability in specific network structures.	MDP, task defined as attack path planning.	Optimizes sample replay by enhancing priority rewards with attack graphs, improving the effectiveness of sparse reward samples.	Tests path planning efficiency and stability on attack graphs of varying scales.	Focuses on reward optimization in attack path planning, suitable for attack strategy design in specific scenarios.

In summary, different methods, through the integration of experience replay mechanisms and deep reinforcement learning, demonstrate unique advantages in addressing challenges such as sparse rewards, large action spaces, and dynamic network environments. From goal redefinition based on HER to sample balancing with expert demonstrations, and further to enhancing adaptability with LSTM and dynamic adjustment modules, these approaches achieve varying degrees of optimization in sample utilization, training efficiency, and environmental adaptability. These studies not only validate the effectiveness of the methods in specific tasks but also provide significant theoretical support and practical references for the field of intelligent penetration testing.

3.20.5 Analysis of Research Related to Hierarchical Mechanisms

From the review, we can also observe that the hierarchical agent approach is widely applied. This includes HA-DRL [6], DynPen [19], HDRL [34], and CRLA [31]. A detailed comparative analysis is shown in Table 3.5.

Table 3.5: Comparison of HA-DRL, HDRL, and DynPen

Feature	**HA-DRL (Tran et al., 2021) [6]**	**HDRL (Li et al., 2023) [34]**	**DynPen (Li et al., 2024) [19]**
Key Characteristics	Uses algebraic decomposition to reduce the large discrete action space.	Combines expert prior knowledge and hierarchical agents to improve sample efficiency and reduce invalid exploration.	Focuses on dynamic network environments, adding adaptability and fast convergence capabilities.
Application Scenarios	Penetration testing in static network environments.	Penetration testing in static or large-scale complex network environments.	Penetration testing in dynamic network environments, with adaptability to scenario changes.
Research Approach	Proposes a hierarchical multi-agent DRL framework to address action space explosion.	Divides tasks into three action types, using expert knowledge to guide action selection and reduce exploration costs.	Introduces trajectory recording and environment monitoring modules in dynamic scenarios to optimize strategy adjustments.
Action Space Hierarchy	Action space is decomposed algebraically, with each subspace assigned to an independent agent.	Action space is divided into three types: local exploitation, remote exploitation, and connection actions, each handled by a specific agent.	Similarly divided into three types but dynamically adjusts action space with scenario adaptation modules.

Feature	HA-DRL (Tran et al., 2021) [6]	HDRL (Li et al., 2023) [34]	DynPen (Li et al., 2024) [19]
Sub-Agent Training	Sub-agents are trained independently and do not share information.	A high-level agent determines which sub-agent to invoke; sub-agents are optimized using expert knowledge.	Sub-agents are trained dynamically and update decision spaces based on real-time monitoring.
Incorporation of Expert Knowledge	Not introduced.	Embeds rules and knowledge graphs to constrain actions and optimize exploration paths.	Does not directly incorporate expert knowledge but indirectly improves decision-making via dynamic adaptation and trajectory records.
Underlying Model	MDP	MDP with partial incorporation of prior knowledge.	MDP, focusing on rapid decision optimization in dynamic scenarios.
Experimental Scenarios	Validated on the CybORG platform, simulating small-scale network environments.	Validated on custom static network scenarios to demonstrate scalability to large networks.	Simulates dynamic network environments to evaluate learning speed and accuracy in dynamic scenarios.
Practicality	Provides a scalable method but requires further validation in large-scale environments.	Significantly reduces sample complexity, making it suitable for large-scale networks.	Offers dynamic adaptability, making it highly applicable in complex real-world environments.

Furthermore, regarding CRLA [31], I also mentioned this in its review. The hierarchical agent structure design of CRLA [31] is essentially identical to that of HA-DRL [6], with only a change in name. Even the segmented tree-related design is the same. The number of hosts and the size of the action space in the experiments are also consistent between the two. The main difference is that CRLA introduces the QMIX mixing network mechanism and applies improvements such as Dueling DQN. Theoretically, it should be able to support larger-scale networks and sparse reward environments.

3.20.6 Other Approaches

In addition to the aforementioned methods, some studies have adopted alternative approaches. Below, we provide an analysis and summary of these studies:

1. **Improved-PenBox** [32]
   This study focuses on a different set of challenges compared to those addressing large network environments, extensive action spaces, or sparse reward settings. Specifically, it combines the original PenBox framework with reinforcement learning to address the issue of requiring fully manual script programming for PenBox operations. This targeted integration makes it a highly specialized solution.

2. **SmartGrid-PTDRL** [7]
   This study is noteworthy as the first and, so far, the only research dedicated to automated penetration testing in smart grids, filling a critical gap in the field. The same author also introduced a novel method called **DRLM-PT**, which employs a reward machine to encode domain knowledge and drive agent behavior. This innovative use of reward machines highlights the author's creative and forward-thinking approach.

3. **INNES** [36]
   This research identifies a significant issue in many studies that rely heavily on prior knowledge, such as expert experience, which can be a limiting factor. INNES addresses this by gradually building a network state representation during the exploration process. It introduces the concepts of a known node set K and a valid action set V, restricting action selection to valid action spaces. This approach reduces dependency on prior knowledge, decreases the size of the action space, and

mitigates the impact of sparse rewards by constraining ineffective exploration. It's a compelling idea that effectively tackles multiple challenges simultaneously.

Chapter 4

Systematic Literature Review of Other Algorithms

In this chapter, we will introduce studies that utilize other algorithms as their core reinforcement learning approach.

4.1 AC (Actor-Critic)

The Actor-Critic (AC) algorithm is a foundational method in reinforcement learning that combines the strengths of value-based and policy-based approaches [38]. It employs two distinct components: the Actor, which learns a policy to select actions, and the Critic, which evaluates the actions by estimating the value function. The Actor updates the policy directly based on feedback from the Critic, while the Critic updates the value function using temporal difference (TD) learning. This synergy allows the AC algorithm to address the inefficiencies of pure policy gradient methods by reducing variance while maintaining stable learning. AC algorithms are flexible and serve as the basis for many advanced reinforcement learning methods, such as A3C (Asynchronous Advantage Actor-Critic) and PPO (Proximal Policy Optimization). They are widely applied in areas like continuous control tasks, financial modeling, and real-time decision-making systems due to their ability to handle high-dimensional action spaces and adapt to dynamic environments.

4.1.1 HAE

In 2022, Hoang Viet Nguyen and Tetsutaro Uehara published a paper titled *"Hierarchical Action Embedding for Effective Autonomous Penetration Test-*

ing" [39]. This paper proposed an automated penetration testing framework based on the Hierarchical Action Embedding (HAE) model. By integrating the MITRE ATT&CK knowledge base and the Wolpertinger architecture (WA) [40], this model efficiently manages and simplifies the large action space, thereby improving the efficiency and accuracy of automated penetration testing. The research aims to address the limitations of traditional reinforcement learning methods in large-scale and complex networks, enhancing the precision and efficiency of automated penetration testing.

The HAE model is designed based on abstraction and encoding of different layers of information in penetration testing. It consists of the following three layers:

Tactical Type Layer

The tactical type layer systematically encodes the various tactics employed by attackers or penetration testers in a network. This layer leverages the MITRE ATT&CK knowledge base, which comprehensively documents the tactics and techniques used by attackers in real-world scenarios. To enable reinforcement learning algorithms to utilize these tactics, the researchers adopted a multi-hot encoding approach. Each tactic is represented by a vector, where each bit corresponds to a specific tactic type. For instance, for a technique associated with a single tactic, such as *Active Scanning (T1595)*, the corresponding bit is set to 1, and all others are 0. For techniques associated with multiple tactics, such as *Valid Accounts (T1078)*, the bits corresponding to multiple tactics are set to 1. This multi-hot encoding not only effectively represents the relationships between techniques and tactics but also ensures that the reinforcement learning model can understand and utilize these relationships to optimize decision-making processes.

Feature Layer

The feature layer refines the representation of techniques in the MITRE ATT&CK knowledge base to generate technical feature embeddings. Here, the researchers used supervised learning methods to vectorize each technique based on its relationships with other security components such as platforms, permissions, malware, attack groups, and mitigations. The core idea is to capture the characteristics of each technique more accurately by analyzing its connections with these components. For example, certain techniques may only be executable on specific platforms or require certain permissions. By modeling and training these relationships, the resulting feature embeddings represent the essence of each technique, enabling reinforcement learning

agents to make more effective decisions in complex network environments.

During the training of technical feature embeddings, the researchers first collected a large dataset of 2 million positive samples from the MITRE ATT&CK knowledge base. They then built a supervised learning model to capture the relationships between techniques and various security components. The model used category embedding to encode these relationships into fixed-size embedding vectors, which were further processed through fully connected layers and cosine similarity calculations to form the feature layer. This layer provides the reinforcement learning model with detailed technical representations, enabling it to distinguish subtle differences between techniques.

Network Structure Layer

The network structure layer focuses on the connections and topology between hosts in the network, providing additional environmental information to the reinforcement learning agent to enhance penetration testing performance. This layer employs Node2Vec technology to transform the entire network topology into n-dimensional vectors representing hosts and their connectivity. The researchers first modeled each host and its associated subnetwork, then used Node2Vec to learn and represent the internal connections of these subnetworks. Node2Vec, a graph embedding technique, simulates random walks to capture node features, generating vector representations for each node. This approach provides the reinforcement learning agent with crucial information about the network topology, such as identifying critical nodes or tightly connected subnetworks. This information helps the agent prioritize scanning important subnetworks or attacking high-value targets, improving testing efficiency and accuracy.

Action Selection Based on Wolpertinger Architecture

After observing the environment state, the framework generates a proto-action. Then, the action embedding layer retrieves the k nearest actions to the proto-action using the kNN algorithm for optimization. Finally, the value evaluation network selects the best action, which interacts with the environment.

Experimental Platform and Environment

To evaluate the effectiveness of the HAE model, the authors built a custom multi-step penetration testing environment. In this environment, specific machines could be exploited using one or more techniques, and penetration

testers had to apply these techniques in a particular sequence to fully compromise target machines. The design of these attack sequences ensured a high degree of randomness and uncertainty, reflecting the challenges of real-world penetration testing. The environment configurations included:

- **Attack Sequences**: Each machine required a specific sequence of techniques to be compromised, with success rates for each step randomly set between 75% and 90%, ensuring realism.
- **Scenario Configurations**: Experiments covered varying numbers of sensitive machines, techniques, and machines to test the HAE model's performance under different levels of complexity.

This environment allowed for flexible control of experimental parameters, enabling comprehensive validation of the proposed method and ensuring that the experimental results accurately reflected the HAE model's performance in networks of varying complexity.

Experimental Validation

The experiments evaluated three different configurations: the number of sensitive machines, the number of techniques, and the number of machines.

- **Sensitive Machines Configuration**: In scenarios with different numbers of sensitive machines, the HAE model outperformed other reinforcement learning algorithms (e.g., DQN, DDQN, A2C), achieving a 100% success rate in compromising sensitive machines in most cases.
- **Techniques Configuration**: As the number of techniques increased, the HAE model maintained a high success rate, while the performance of DQN-based algorithms significantly declined.
- **Machines Configuration**: In larger network environments, the HAE model demonstrated stability, showcasing its superiority in handling complex network structures.

4.1.2 MAR-WA

In 2023, Nguyen and Uehara published a paper titled *"Multilayer Action Representation based on MITRE ATT&CK for Automated Penetration Testing"* [12]. The paper proposed a Multilayer Action Representation (MAR) based on the MITRE ATT&CK framework, aiming to enhance the efficiency and

accuracy of reinforcement learning (RL) in complex network environments. By representing penetration testing actions as n-dimensional vectors, MAR enables RL algorithms to handle penetration testing tasks in large, complex networks more effectively.

MAR-WA: Structuring Penetration Testing Actions into Three Layers

1. **Action Type Layer**: Based on MITRE ATT&CK tactics, this layer describes the high-level goals of penetration testing actions, such as privilege escalation, persistence, and lateral movement. Using multi-hot encoding, this layer helps the agent identify logical relationships between different actions.

2. **Action Technique Layer**: This layer captures specific attack techniques used in penetration testing. Techniques are represented based on their relationships with other components in the MITRE ATT&CK framework, such as platforms, privileges, and software. This helps RL agents understand similarities between techniques and apply them in complex network attacks.

3. **Action Object Layer**: This layer represents the targets of penetration testing using network topology information, such as subnets and hosts. Techniques like Node2Vec are employed to embed network nodes (e.g., hosts and routers) as vectors, aiding the agent in understanding attack paths and relationships between different hosts in the network.

By integrating these layers, MAR accurately represents the action space in large-scale networks, improving the learning efficiency of agents and enhancing the effectiveness of their attack strategies.

As shown in 4.1, the implementation of the Technique Feature layer is based on the relationships between attack techniques and other components in the MITRE ATT&CK framework. This layer uses a neural network model to embed techniques into vectors, allowing the agent to understand the similarities and differences between various attack techniques. The model takes into account features such as platforms, privileges, malware, groups, and mitigations to create a comprehensive representation of each technique.

Epsilon-Wolpertinger Architecture

To address the challenges of high-dimensional discrete action spaces in complex networks, the authors introduced an epsilon-Wolpertinger architecture

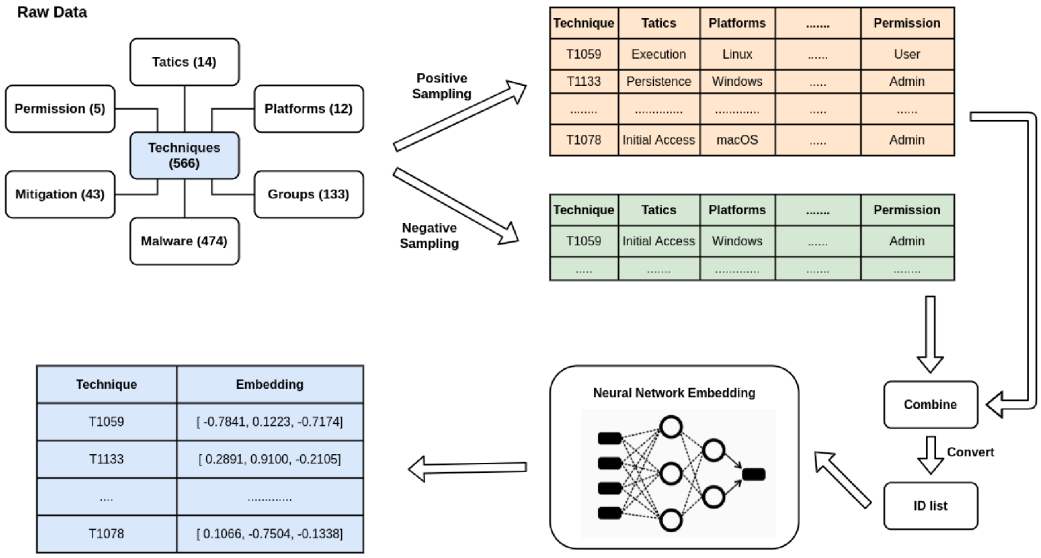


Figure 4.1: Implementation of Technique Feature layer [12].

[40] (Gabriel Dulac-Arnold et al., 2015) based on k-nearest neighbors (kNN) and reinforcement learning. This architecture optimizes training time by reducing unnecessary action searches. In the early training phase, epsilon-Wolpertinger explores unoptimized action spaces and gradually converges to more precise action selections, significantly reducing training complexity.

As shown in 4.2, the Wolpertinger architecture combines action embedding with deep reinforcement learning to handle large discrete action spaces effectively. The architecture uses a proto action selected by the actor network and k secondary actions chosen via the kNN algorithm to improve the agent's ability to select optimal actions. The critic network evaluates the actions, and the argmax function determines the best action to take in each step.

Experimental Validation

The authors conducted various experiments to validate MAR's effectiveness, including different network configurations, the number of sensitive hosts, types of techniques, and device counts. They designed scenarios with four difficulty levels (easy, normal, medium, and hard) and compared MAR with several popular RL algorithms, such as DQN, Double DQN, and A2C.

- **Easy and Normal Difficulty Levels**: MAR exhibited a very high success rate, significantly outperforming traditional algorithms when attacking multiple sensitive hosts and using more attack techniques.

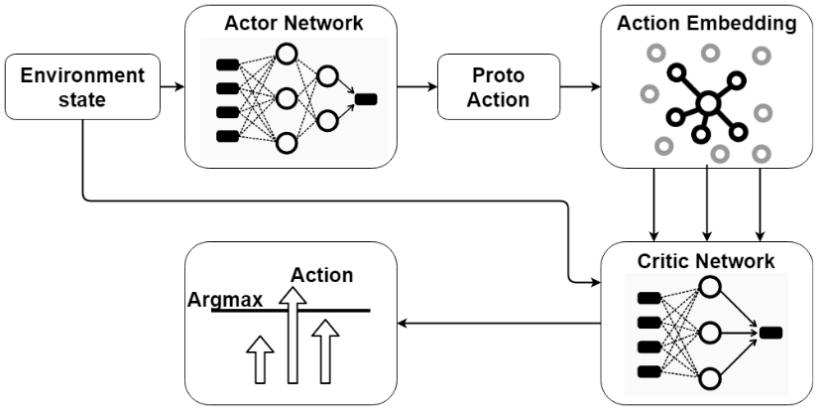


Figure 4.2: Wolpertinger architecture [12].

- **Medium and Hard Difficulty Levels**: As attack sequences grew longer and the success rates of techniques decreased, the performance of traditional algorithms declined, whereas MAR maintained robust performance. In multi-stage, multi-technique environments, particularly in large-scale network systems, MAR demonstrated stability and effectiveness.

4.1.3 MLAE-WA

In 2020, Nguyen et al. published a paper titled *"Multiple Level Action Embedding for Penetration Testing"* [41], which mainly addresses the huge action space problem faced by reinforcement learning algorithms in penetration testing in complex large-scale network environments. In order to improve the efficiency and accuracy of these algorithms, the authors proposed a multi-level action embedding method combined with the Wolpertinger architecture (WA) [40] to optimize the reinforcement learning process for penetration testing. The main contribution is that the method of the WA architecture is introduced into the field of automatic penetration testing.

Research Methodology

The core of the study lies in the multilayer action embedding approach, which facilitates better understanding and selection of actions by reinforcement learning algorithms. This approach includes three layers of embeddings:

1. **Action Feature Embedding**: Focuses on the type and target of actions. For example, action types may include subnet scanning, host

scanning, or exploiting service vulnerabilities, while targets are specific subnets or hosts.

2. **Network Structure Embedding**: Uses the Node2Vec model to embed the network topology, representing relationships between subnets and hosts. This embedding helps the algorithm understand the connections between nodes in the network, optimizing the selection of attack paths.

3. **Service Vulnerability Embedding**: Embeds information about the services running on hosts and their vulnerabilities. By processing vulnerability descriptions (e.g., CVE identifiers and details) with neural networks, the approach generates n-dimensional vector representations of services. This embedding enables the algorithm to better assess which services are more susceptible to attack.

These embedding layers work in tandem to help reinforcement learning algorithms select the most appropriate attack actions, thereby improving performance in complex network environments.

Experimental Design

To validate the effectiveness of the multilayer action embedding, the authors conducted multiple experiments, divided into the following parts:

1. **Action Embedding Testing**: The authors first tested the logical accuracy of the proposed multilayer action embedding approach. The experiments verified whether the embedding layers accurately reflected the relationships between different attack actions. Specifically, in the service vulnerability embedding, the method demonstrated the ability to group similar services based on vulnerabilities accurately.

2. **Performance Testing of Wolpertinger Architecture with Multilayer Embedding**: In this experiment, the authors combined the multilayer action embedding with the Wolpertinger architecture for penetration testing and compared it with the traditional DQN algorithm. They tested the system's performance under scenarios of varying complexity:

    - *Proportion of Sensitive Hosts*: By increasing the proportion of sensitive hosts in the network, the difficulty of the task was raised.

The results showed that as the proportion of sensitive hosts increased, the performance of the traditional DQN algorithm decreased. In contrast, the WA combined with multilayer embedding maintained over 90% accuracy even when the proportion of sensitive hosts reached 100%.

- *Service Success Rate*: The algorithm's performance was tested in more challenging environments by lowering the success rate of service attacks. The results revealed that when the service success rate dropped to 20%, the accuracy of the DQN algorithm decreased significantly, whereas the WA with embedding maintained over 90% accuracy.
- *Network Scale Testing*: In large-scale network environments (with host counts ranging from 10 to 256), the experiments demonstrated that as the network scale reached 64 hosts, the performance of the DQN algorithm nearly dropped to zero. However, the WA combined with multilayer embedding maintained approximately 70% accuracy even in networks with 256 hosts.

Experimental Results

The experimental results confirmed the effectiveness of the multilayer action embedding approach. In complex network topologies and service vulnerability environments, this method enabled reinforcement learning algorithms to more accurately select optimal attack paths. Additionally, the algorithm significantly outperformed traditional methods in large-scale network environments.

By introducing multilayer action embedding, Nguyen et al. substantially improved the applicability of reinforcement learning algorithms in penetration testing. This approach not only expanded the network scales manageable by the algorithms but also enhanced their attack success rates in complex environments.

4.1.4 Double Agent Architecture (DAA)

In 2021, Nguyen et al. published a paper titled *The Proposal of Double Agent Architecture using Actor-Critic Algorithm for Penetration Testing* [42], which introduced a Double Agent Architecture (DAA) that applies reinforcement learning to large-scale networks (up to 1,000 nodes) while ensuring performance stability.

Research Methodology

1. Improved Network Attack Simulator (NAS) The authors first modified the original Network Attack Simulator (NAS) and proposed an enhanced version by adding new rules to increase complexity:

- **Subnet level**: Additional constraints were introduced, such as preventing attackers from knowing connections between subnets at the start; attackers must gradually discover this information through scanning.
- **Host level**: The concept of sensitive hosts was introduced, which contain critical data and offer higher attack rewards, encouraging agents to prioritize key targets.
- **Service level**: Each service was assigned a randomly distributed success rate and attack cost, which dynamically adjust to further increase the simulation environment's realism.
- **Rewards**: Several modifications were made to address the sparse reward issue, detailed in the rewards section below.

2. Proposed Double Agent Architecture

- **Structural Agent**: Responsible for exploring and identifying the overall network architecture, including scanning subnets and hosts to gather network information.
  - *Action space*: Scanning subnets, scanning hosts, and passing host information to the exploitation agent.
  - *State space*: Includes only the reachability and attack status of hosts within the network, significantly reducing complexity.
- **Exploitation Agent**: Focused on selecting appropriate services to attack on selected hosts while also refining target host information through scanning.
  - *Action space*: Scanning services and exploiting services on specific hosts.
  - *State space*: Concentrates on the information of target hosts and their services to enable more targeted decision-making.

By dividing action and state spaces logically, the Double Agent Architecture significantly reduces the computational complexity each agent needs to handle, making reinforcement learning suitable for larger networks.

3. Reinforcement Learning Algorithm

- **A2C Algorithm in DAA Architecture**: The authors selected the Advantage Actor-Critic (A2C) algorithm for reinforcement learning due to its efficiency in learning complex policies in asynchronous parallel environments and its good convergence and flexibility.

- **Baseline**: The Wolpertinger architecture was used as a baseline model, as it addresses the large action space problem through action embedding and k-Nearest Neighbor (kNN) search.

The authors used two environments overall:

- The improved NAS was used to train other agents, including the baseline WA and the DQN series (DQN, DDQN, Dueling DQN, and D3QN).

- A separate environment was designed for the Double Agent Architecture, based on the improved NAS but with modifications to match the DAA architecture, including adjusted action spaces and a reward mechanism tailored for the DAA.

Reward Mechanism

1. Reward Mechanism in the Improved NAS

- **Reward Calculation**: Rewards are based on the success of host attacks. Successfully attacking a normal host yields a base reward r. For sensitive hosts, the reward is adjusted proportionally based on the ratio of normal to sensitive hosts, calculated as $(H_{\text{normal}}/H_{\text{sensitive}}) \times r$.

- **Penalty Mechanism**: Any action that changes the environment state incurs a small penalty to discourage ineffective or resource-wasting behavior.

- **Objective**: This mechanism encourages agents to expand their attack range and prioritize sensitive hosts while maintaining efficiency through penalties.

2. Reward Mechanism in DAA

- **Structural Agent Rewards**:

  – Operations like scanning subnets or hosts provide an immediate base reward r_1.

  - If the structural agent selects the "choose host" action, its reward includes the exploitation agent's total reward (i.e., $r_1 + r_2$).

- **Exploitation Agent Rewards**:
  - Rewards for operations like attacking services are directly based on the success of exploitation, following a similar mechanism as in NAS.

- **Design Characteristics**:
  - The reward mechanism separates network exploration (handled by the structural agent) and host exploitation (handled by the exploitation agent).
  - This separation reduces the state and action space complexity for each agent, improving efficiency in large network environments.

- **Objective**: The DAA reward mechanism aims to guide the structural agent to explore network topology while enabling the exploitation agent to focus on attacking optimal services, enhancing success rates in complex network environments.

Experiments

The experiments consisted of two parts to test algorithm performance in small and large network environments.

1. Small Network Environment

- Network size ranged from 5 to 50 hosts, each running 2 services, with a fixed number of 7 subnets. The number of sensitive hosts was dynamically adjusted based on total hosts.

- Results showed that when the number of hosts was below 20, all algorithms achieved over 90% attack success rates on sensitive hosts. However, when the host count exceeded 20, the performance of DQN series algorithms rapidly declined.

- In contrast, DAA and A2C with the Wolpertinger architecture maintained high success rates in most scenarios, with DAA achieving a 70% success rate in a 50-host environment, while A2C reached nearly 99%.

2. Large Network Environment

- The number of hosts was expanded to between 64 and 1,024, each running 5 services, with 20 fixed subnets and sensitive hosts accounting for 10% of normal hosts.
- Results showed that the Wolpertinger architecture with A2C failed completely when the host count reached 512 or more, with a 0% success rate on sensitive hosts.
- In contrast, DAA maintained an 81% success rate under the same conditions. Even when the number of services per host increased to 100, DAA still achieved a 70% success rate.
- Other algorithms' performance drastically declined as the number of hosts and services increased.

The results clearly demonstrate that DAA's Double Agent Architecture effectively reduces the complexity of state and action spaces for each agent, significantly improving efficiency and stability in performing penetration testing tasks in complex, large-scale network environments. Compared to other algorithms, DAA exhibited superior scalability and adaptability.

4.1.5 Overall Analysis of AC Approaches

From the previous review, we can see that there are currently four studies based on the Actor-Critic (AC) method, all of which were conducted by the same research team. These studies demonstrate a gradual evolution from theoretical exploration to practical application, encompassing various methodological improvements and experimental setups. A detailed comparative analysis is presented in the Table 4.1.

Table 4.1: Comparison of MALE-WA, DAA, HAE, and MAR-WA

Characteristic	**MALE-WA (2020) [41]**	**DAA (2021) [42]**	**HAE (2022) [39]**	**MAR-WA (2023) [12]**
Core Method	Multiple Level Action Embedding (MLAE) combined with Wolpertinger Architecture (WA)	Double Agent Architecture (DAA) combined with A2C algorithm	Hierarchical Action Embedding (HAE) combined with MITRE ATT&CK knowledge base	Multilayer Action Representation (MAR) combined with epsilon-WA
Application Domain	Action optimization in penetration testing	Large-scale penetration testing in complex networks	Enhancing RL practicality and efficiency in complex networks	Improving automated penetration testing in complex networks through multilayer representation
Research Method	Based on Markov Decision Process (MDP)	Based on MDP	Based on MDP	Based on MDP
Experimental Environment	Tested up to 256 hosts with limited complexity	Tested up to 1000 hosts with enhanced efficiency through dual agents	Multi-step environment tested, supporting ATT&CK knowledge integration	Tested 119 complex scenarios covering different difficulty levels and network configurations
Key Features	Focuses on reducing action space through embedded vector representation	Dual-agent architecture: one learns network structure, the other selects exploitable services	Combines hierarchical embedding with ATT&CK to improve action space representation and efficiency	Optimizes RL performance in complex networks using ATT&CK and multilayer embedding, integrated with epsilon-WA to reduce training time
Experimental Results	WA+MLAE performed well in complex environments, maintaining accuracy above 90%	Compared with DQN family algorithms, DAA achieved the best performance with 81% accuracy on networks of 1000 hosts	Compared with four RL algorithms, HAE significantly improved efficiency and accuracy	epsilon-WA significantly shortened training time; MAR outperformed all compared algorithms in complex networks

Characteristic	MALE-WA (2020) [41]	DAA (2021) [42]	HAE (2022) [39]	MAR-WA (2023) [12]
Practicality	Initially verified the potential of WA in penetration testing	Extended penetration testing to large-scale networks	Closer to real-world multi-step scenarios, enhancing practical applications with ATT&CK integration	Improved potential for practical penetration testing tools, especially in complex and multi-step environments

In conclusion, this author's research has gradually progressed towards larger-scale and more complex network penetration testing environments. It has evolved from early action embedding to more sophisticated hierarchical and multilayer action representation methods. The introduction of the MITRE ATT&CK knowledge base marks a significant turning point in this research, making the transition from theoretical exploration to practical application more evident. The experiments have gradually expanded from simple simulated environments to multi-scenario, multi-step complex simulations, demonstrating the practical scalability of the technology.

4.2 PPO (Proximal Policy Optimization)

The PPO (Proximal Policy Optimization) algorithm was introduced in the paper *"Proximal Policy Optimization Algorithms"* [43]. It is a policy-gradient-based reinforcement learning algorithm designed to optimize policies in an efficient and stable manner. PPO improves upon earlier policy gradient methods by using a clipped objective function that prevents large updates to the policy, thus maintaining training stability. It also employs a surrogate loss function that ensures the updated policy stays close to the old policy by constraining the policy ratio within a trust region. PPO uses multiple epochs of minibatch updates and leverages advantages from the Generalized Advantage Estimation (GAE) technique to reduce variance in the policy updates. Due to its simplicity, computational efficiency, and high performance, PPO has become one of the most popular reinforcement learning algorithms and is widely applied in areas such as robotic control, game playing, and autonomous navigation.

4.2.1 CLAP

In 2022, Yang Yizhou and Liu Xin published a paper titled *"Behaviour-Diverse Automatic Penetration Testing: A Curiosity-Driven Multi-Objective Deep Reinforcement Learning Approach"* [13]. The paper proposed an innovative automated penetration testing framework called CLAP (Curiosity-Driven Multi-Objective Deep Reinforcement Learning Approach). It primarily addresses the limitations of existing methods regarding single-objective focus and dynamic action space challenges.

The main challenge tackled by the paper is the need for more comprehensive attack strategies in automated penetration testing. Existing approaches often focus on a single objective, such as finding a specific attack path to compromise a target. However, in real-world penetration testing, attackers

must balance multiple, often conflicting objectives, such as maximizing system exploitation while minimizing time or resource consumption. Therefore, automated penetration testing agents must not only identify effective attack paths but also generate diverse behaviors to enable more thorough security assessments.

Methodology CLAP framework integrates multi-objective optimization, coverage masking mechanism, curiosity-driven rewards, and Proximal Policy Optimization (PPO) to generate diverse attack strategies for automated penetration testing. The detailed components are described as follows:

Multi-Objective Reinforcement Learning. CLAP formulates penetration testing as a Multi-Objective Markov Decision Process (MOMDP), represented as a tuple $\langle S, A, P, \vec{R}, \gamma \rangle$, where S is the state space, A is the action space, P is the state transition probability, $\vec{R}$ is the vectorized reward function, and γ is the discount factor. Each objective, such as success rate of exploiting vulnerabilities and operational time cost, corresponds to a dimension in the reward vector. To handle the conflicts between multiple objectives, CLAP uses Chebyshev scalarization to convert the multi-dimensional reward vector into a scalar value [13]:

$$V_{\text{cheby}}(S) = \max_{o=1,\ldots,n} w_o \cdot |V_o(S;\omega) - Z_o^*|, \tag{4.1}$$

where w_o is the relative weight of the o-th objective, and Z_o^* represents the ideal value for the objective. This approach generates a set of Pareto optimal solutions, effectively balancing the potentially conflicting goals.

Coverage Masking Mechanism. To improve exploration efficiency and avoid repeated operations, CLAP introduces a coverage masking mechanism that tracks historical actions and dynamically shifts focus to unexplored areas of the network. A coverage vector C_t is maintained to record the agent's past actions [13]:

$$C_t = \sum_{t'=0}^{t-1} A_c^{t'}, \tag{4.2}$$

where $A_c^{t'}$ represents the one-hot embedding of the action taken at time step t'. This vector is fused with the agent's current policy output to dynamically adjust the exploration strategy. The fusion weights are adaptively generated based on the current state, ensuring flexibility in different network environments.

Curiosity-Driven Rewards. To address the issue of sparse rewards in penetration testing, CLAP employs Random Network Distillation (RND) to generate intrinsic rewards. RND consists of a target network and a predictor

network, with the target network providing fixed features and the predictor network attempting to predict them. The prediction error is used as the intrinsic reward [13]:

$$R_{\text{curiosity}} = \|f_{\text{target}}(s) - f_{\text{predictor}}(s)\|^2, \tag{4.3}$$

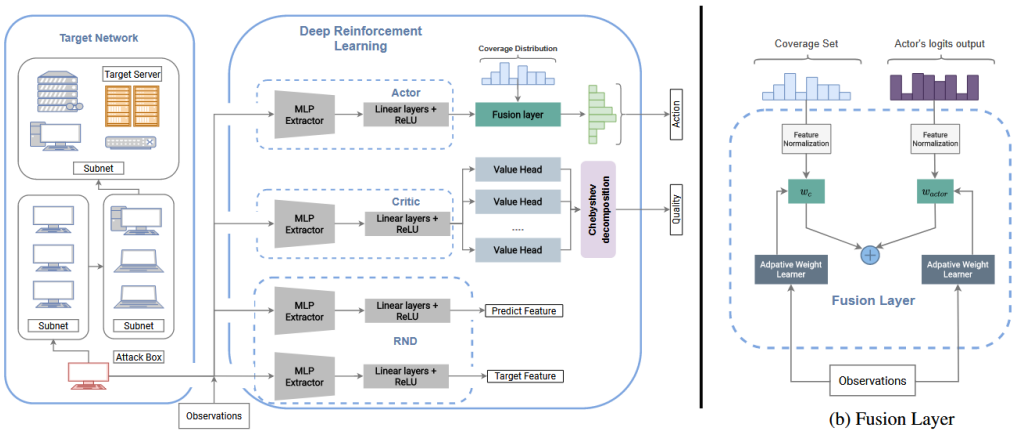
where f_{target} and $f_{\text{predictor}}$ are the outputs of the target and predictor networks, respectively. This mechanism drives the agent to explore unknown regions even in the absence of external rewards.

Proximal Policy Optimization. CLAP utilizes Proximal Policy Optimization (PPO), a robust policy optimization algorithm, to ensure stable and efficient training. The PPO objective function is defined as [13]:

$$J_{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right], \tag{4.4}$$

where $r_t(\theta)$ is the probability ratio between the new and old policies, A_t is the advantage function, and ϵ is a hyperparameter that limits the update step size. By constraining policy updates, PPO enhances the training stability and convergence speed, making it well-suited for dynamic penetration testing scenarios.

CLAP framework effectively combines multi-objective optimization, dynamic exploration mechanisms, intrinsic rewards, and stable training techniques, enabling the generation of diverse and efficient attack strategies in complex and large-scale penetration testing environments.



(a) System Architecture

(b) Fusion Layer

Figure 4.3: Architecture of CLAP [13].

Experimental Evaluation To validate the proposed method, the paper conducted experiments in multiple benchmark scenarios, including NAS (Network Attack Simulator) and a modified version of the 2021 IJCAI CAGE Challenge. These experiments evaluated the multi-objective learning capabilities and performance efficiency of the CLAP framework. Results demonstrated that the framework outperformed existing reinforcement learning algorithms in generating diverse attack strategies and handling large-scale networks. Notably, it exhibited significant stability and effectiveness in complex network environments.

4.2.2 EPPTA

In 2023, Li Zegang and colleagues published a paper titled *"EPPTA: Efficient Partially Observable Reinforcement Learning Agent for Penetration Testing Applications"* [14]. This paper proposed an automated penetration testing framework based on reinforcement learning, aimed at addressing challenges in partially observable environments to improve testing efficiency and accuracy.

Framework Design and Principles

The core of the framework lies in applying the Partially Observable Markov Decision Process (POMDP) to reinforcement learning. This framework, named **EPPTA** (Efficient Partially Observable Markov Decision Process-driven Penetration Testing Agent), is structured around the following key stages:

Construction and Optimization of the Implicit Belief Module The belief module is the innovative core of EPPTA. By integrating the traditional POMDP belief update formula with deep neural networks, it dynamically predicts and adjusts the agent's probability distribution over the environment's states. The belief state update process has two stages:

- **Stage 1:** Utilize Long Short-Term Memory (LSTM) networks to process historical observations (i.e., action and observation sequences) and generate temporary belief states.
- **Stage 2:** Adjust the belief state using observation probabilities and Bayes' theorem to more accurately reflect the dynamic changes in the current environment.

The paper introduced a matrix-based belief representation method to significantly reduce computational complexity in high-dimensional belief dis-

tributions. Multi-Layer Perceptrons (MLPs) are employed to approximate the update process, making belief state updates more efficient and adaptive.

Implementation and Optimization of the Asynchronous Reinforcement Learning Framework To address communication and computational bottlenecks in large-scale parallel environments, EPPTA employs a highly optimized asynchronous reinforcement learning process through the Sample Factory framework. Specifically:

- The training process is divided into three independent modules: **Rollout Workers** (environment interactions), **Policy Workers** (policy inference), and **Learner** (parameter updates). These modules achieve efficient asynchronous communication through shared memory and signaling mechanisms.
- To reduce data transmission overhead, key data (e.g., belief states) are stored in shared GPU memory, and a double-buffered sampling strategy ensures the continuity and consistency of interaction data across multiple environments.
- By dynamically adjusting the buffer time window to accommodate interaction time differences across environments, the framework significantly improves data sampling efficiency and avoids performance bottlenecks caused by uneven workloads.

Policy Optimization and Loss Function Design EPPTA incorporates a reinforcement learning objective function based on the Proximal Policy Optimization (PPO) algorithm and introduces a **Belief Loss** to guide the agent in learning strategies more effectively in partially observable environments. The loss function considers both the magnitude of changes between the current and previous policies and introduces a belief consistency term to constrain the rationality of belief updates. This ensures both stability in policy updates and improved accuracy in predicting environmental states during training.

As shown in Figure 4.4, the upper part illustrates an overview of the POMDP, while the lower part describes the EPPTA flowchart. An innovative belief module is introduced, which collaborates with the Actor-Critic network framework to implicitly comprehend the belief update for the penetration testing environment state [14].

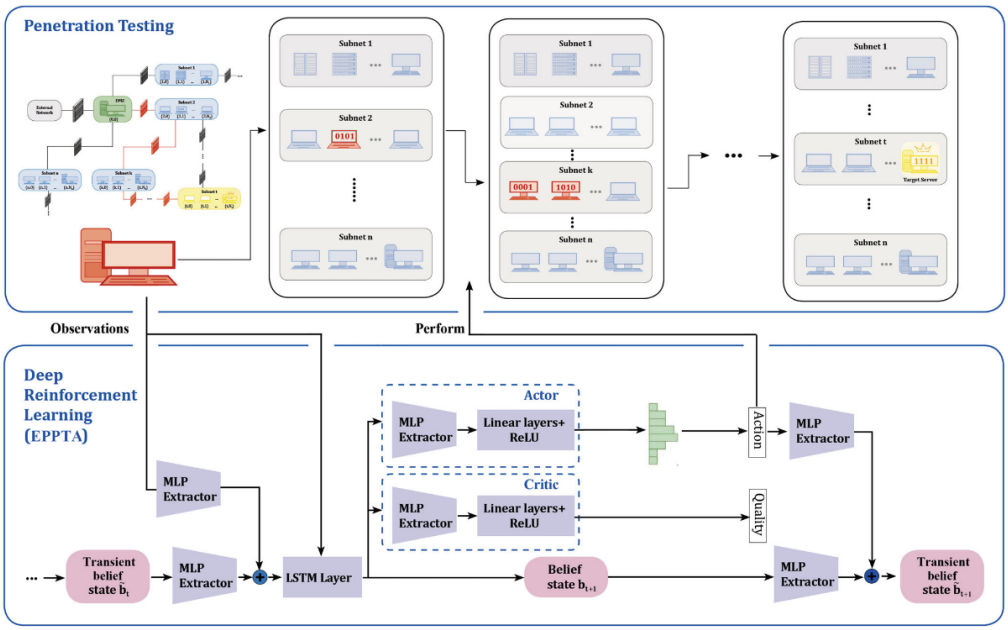


Figure 4.4: Schematic representation of the EPPTA [14].

Experiments

The EPPTA framework was systematically evaluated on the NAS simulation platform. The experimental process included the following critical components, each rigorously designed and measured with detailed performance metrics:

1. Environment Configuration and Experimental Scenarios The experiments utilized a range of network configurations from simple to complex, including six scenarios: *Tiny*, *Small*, *Medium*, *LargeGen*, *HugeGen*, and *Pocp2Gen*. Each scenario defined distinct network topologies, numbers of subnets, hosts, and dimensions of state and action spaces. For instance, the *Pocp2Gen* scenario includes 21 subnets, 95 hosts, an action space dimension of 3,515, and a state space dimension of 1.49 billion. The goal in these scenarios was to control all target hosts as quickly as possible while minimizing action costs.

2. Baselines and Metric Selection To comprehensively evaluate EPPTA's performance, multiple existing baseline methods were included, such as *HA-DQN*, *NDSPI-DQN*, PPO models, and Recurrent PPO models, representing traditional deep Q-learning, improved reinforcement learning algorithms, and

recurrent policy networks for penetration testing. Metrics included:

- **Interquartile Mean (IQM)** rewards and step lengths to measure the efficiency and convergence of the agent's strategies in partially observable environments.
- **Gap to theoretical optimal value (Oracle)** to assess the algorithm's performance relative to expert strategies.
- **Convergence time and training efficiency** to evaluate the scalability and resource utilization of the algorithm in different scenarios.

3. Experimental Design and Performance Validation

- **Performance comparison in standard scenarios:** All algorithms were tested across scenarios, and their rewards and convergence speeds were recorded. EPPTA achieved the highest rewards in large-scale scenarios (e.g., *Pocp2Gen*) while significantly reducing the steps needed to complete tasks. Its reward values closely approached the theoretical optimal, with the smallest gap to the Oracle.
- **Random node isolation experiments:** To verify EPPTA's robustness in dynamic network environments, a random node isolation mechanism was introduced to simulate the defensive behavior of network administrators (randomly isolating some nodes). Results showed that EPPTA maintained high rewards and low step lengths even with isolation probabilities as high as 70%, while other algorithms experienced significant performance degradation.
- **Convergence time and scalability analysis:** By leveraging the high-performance asynchronous architecture, EPPTA demonstrated significantly faster convergence times compared to other methods across scenarios.

4. In-depth Analysis and Ablation Experiments To explore the contribution of EPPTA's components, experiments systematically removed core elements such as the belief module, double-buffered sampling strategy, and asynchronous architecture. Results indicated that the belief module provided the most significant reward improvements in partially observable environments, while the double-buffered sampling strategy enhanced data utilization efficiency and training stability.

4.2.3 RLAPT

In 2023, Xiaotong Guo and colleagues published a paper titled *"Automated Penetration Testing with Fine-Grained Control through Deep Reinforcement Learning"* [15]. This paper proposed an innovative automated penetration testing framework named RLAPT (Reinforcement Learning Automated Penetration Testing), where the penetration testing problem was modeled as a Partially Observable Markov Decision Process (POMDP).

Details of RLAPT

Action Decomposition To address the challenge of managing large-scale action spaces, RLAPT employs an action decomposition strategy, dividing the complex action space into five smaller sub-action spaces. When faced with complex attack decisions, this strategy allows the agent to progressively choose from a series of relatively simple sub-actions, thereby learning fine-grained control over attack actions. This decomposition significantly reduces the number of options the agent must consider at each decision step, making large and complex action spaces more manageable and easier to learn. Additionally, the decomposed sub-action spaces enable the agent to optimize independently within each space, improving learning speed and accuracy.

Action Masking RLAPT introduces action masking to further enhance the training efficiency and stability of the agent. Action masks filter out invalid attack actions in the current state. For example, certain attack types can only be executed under specific conditions, or further attacks can only be conducted if a specific node has already been compromised. By using action masks, RLAPT significantly reduces the number of invalid actions attempted by the agent, concentrating resources on actions more likely to succeed, thus improving exploration efficiency.

Probability Masking Probability masking is used to exclude attack parameters irrelevant to the current attack type. Different attack types may require different parameters. For instance, remote attacks require selecting a target node and a remote vulnerability, while local attacks only require selecting a local vulnerability. By applying probability masks during training, RLAPT effectively reduces training noise and avoids interference from irrelevant parameters. This approach ensures the agent focuses on the most relevant information during training, reducing training time and improving final convergence performance.

Reward Function The authors designed a direct reward assignment strategy for different attack outcomes. For example:

- Discovering a new node: +5 points
- Discovering a new credential: +3 points
- Failed attack: -1 point

The reward function is designed to encourage the agent to conduct more effective exploration and attacks.

Deep Reinforcement Learning Algorithm

RLAPT adopts an Actor-Critic architecture, specifically utilizing the Proximal Policy Optimization (PPO) algorithm. This architecture combines the advantages of policy learning and value evaluation:

- **Actor:** Learns how to select optimal actions for given states.
- **Critic:** Evaluates the value of those actions.

This dual functionality allows the agent to simultaneously improve its policy and evaluate value, maintaining directionality during the learning process. PPO further ensures stability by limiting the step size of policy updates, balancing exploration and exploitation, and preventing excessive policy changes during updates.

As shown in Figure 4.5, the RLAPT architecture integrates the observation space, action space, reward signal, and training process. The observation space is designed to provide the agent with partial but incrementally updated information about the network, while the action space is decomposed into multiple sub-action spaces to manage the complexity of attack decisions. The action and probability masking mechanisms further refine the agent's decision-making process by filtering out invalid and irrelevant actions. The Actor-Critic architecture, guided by the PPO algorithm, ensures efficient and stable training. This comprehensive design enables RLAPT to achieve fine-grained control over the automated penetration testing process, demonstrating superior performance in both learning speed and convergence stability.

Experiments

RLAPT was validated through simulated experiments on Microsoft's CyberBattleSim platform. CyberBattleSim provides a high-level abstraction

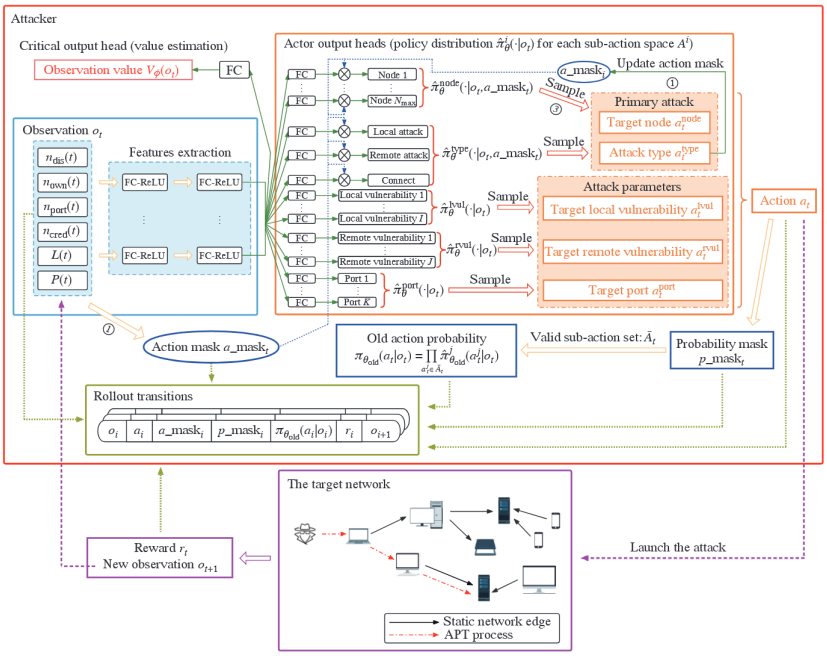


Figure 4.5: Architecture of RLAPT [15].

of computer networks, enabling the RLAPT agent to explore and attack a virtual network with predefined vulnerabilities. The experimental results showed that RLAPT performed exceptionally well in the simulated network, achieving high convergence efficiency and effectively mastering the automated penetration testing process.

Comparisons with Baseline Agents RLAPT was compared against several baseline agents to evaluate its performance:

1. **Traditional Random Attacker:** A baseline that randomly selects valid attack actions for penetration testing.

2. **Multi-Agent Reinforcement Learning (MARL) Attacker:** Employing a hierarchical reinforcement learning framework, where multiple agents manage different sub-action spaces.

3. **DQN Vulnerability Exploiter:** A method implemented by Microsoft on the CyberBattleSim platform. The agent selects vulnerabilities or ports based on current observations.

4. **DRL Primary Attack Deciders:** Agents using DQN, A2C, and PPO algorithms, implemented by the authors using open-source code

and frameworks (details not fully described in the paper).

Results The experimental results demonstrated that RLAPT significantly outperformed these baseline agents in terms of learning speed and stability. RLAPT reduced convergence time by approximately 40%, enabling the agent to find the optimal attack strategy much faster.

Additionally, RLAPT was tested under different knowledge scales to verify its adaptability. Regardless of network scale or complexity, RLAPT effectively learned attack strategies. This flexibility makes RLAPT highly suitable for diverse penetration testing scenarios, allowing it to efficiently learn attack strategies even with limited initial knowledge of the network.

4.2.4 Overall Analysis of PPO Approaches

From the above review, it can be observed that PPO (Proximal Policy Optimization) has been applied in reinforcement learning-based penetration testing research due to its stability and efficiency in handling complex tasks. These studies demonstrate unique advantages in different application scenarios and technical implementations. To comprehensively understand their characteristics, application domains, and methodologies, a detailed comparative analysis of these three studies is provided in the Table 4.2.

Table 4.2: Comparison of CRLA, EPPTA, and RLAPT

Feature	**CRLA (Khuong Tran et al., 2022) [33]**	**EPPTA (Li et al., 2023) [14]**	**RLAPT (Guo et al., 2023) [15]**
Research Method	Multi-objective deep reinforcement learning: Combines Random Network Distillation (RND) and Chebyshev decomposition to resolve multi-objective conflicts and generate diverse attack strategies.	Asynchronous reinforcement learning: Integrates an implicit belief module with the Sample Factory framework for efficient training and rapid convergence, suitable for large-scale networks.	PPO-based deep reinforcement learning: Employs multi-discrete action space decomposition and probability masking to optimize partial observability and improve action selection efficiency.
Application Scenarios	Simulates multi-objective automated penetration testing, suitable for dynamic networks with partial observability.	Penetration testing in partially observable networks, particularly for large-scale networks.	Designed for penetration testing in complex large-scale network environments, addressing partial observability issues.
State Modeling	MOMDP: States include the topology of the discovered network and information about known hosts.	POMDP: States represent belief updates, combining observations with implicit beliefs.	POMDP: States represent limited observations and inferences about the target network.
Action Space	Dynamic, growing discrete action space; employs a coverage masking mechanism to reduce search complexity.	Fixed action space; uses a belief module to predict state probability distributions.	Multi-discrete action space design; adopts decomposition and masking strategies to handle large action spaces.

Feature	CRLA (Khuong Tran et al., 2022) [33]	EPPTA (Li et al., 2023) [14]	RLAPT (Guo et al., 2023) [15]
Reward Mechanism	Vectorized rewards for multi-objective optimization (e.g., balancing time efficiency and vulnerability exploitation).	Reward function incorporates belief state updates to enhance the precision of attack strategies.	Reward function integrates partial observability and exploration of actions.
Experimental Platform	Simulated penetration testing scenarios to verify multi-objective learning and efficiency.	NAS environment, demonstrating high efficiency across different network scales.	CyberBattleSim platform, verifying stability and efficiency in complex network scenarios.
Learning Efficiency	Proposes coverage masking to reduce action redundancy and improve strategy exploration efficiency.	Achieves a 20x training speed-up through an asynchronous framework optimizing data collection and training.	Implements action decomposition and masking strategies for efficient training, reducing redundant actions.
Scalability	Supports dynamically expanding action spaces, adapting to large-scale networks.	Suitable for ultra-large-scale partially observable environments, supporting parallel training.	Adapts to diverse and dynamic target networks through fine-grained control and action decomposition.
Practicality	Suitable for multi-objective optimization in penetration testing, providing diverse security assessments.	Emphasizes rapid convergence and precise strategies in partially observable environments.	Focuses on stable attack strategy design in large-scale networks.

In summary, the three studies each have unique features in terms of technical implementation and application scenarios. CRLA, by integrating stochastic network distillation and multi-objective optimization methods, is suitable for multi-objective penetration testing scenarios that require diversified strategies. EPPTA, centered on asynchronous reinforcement learning and belief modules, performs exceptionally well when dealing with large-scale partially observable networks. Meanwhile, RLAPT emphasizes fine-grained control and stable learning in complex network environments through action space decomposition and masking strategies. These studies demonstrate that reinforcement learning methods based on PPO have significant advantages in the field of penetration testing and allow for the selection of appropriate technical solutions based on specific task requirements.

4.3 Other Approaches

4.3.1 SeqGAN-PT

In 2023, Ankur Chowdhary and colleagues published a paper titled *"Generative Adversarial Network (GAN)-Based Autonomous Penetration Testing for Web Applications"* [44], proposing a novel method for automating penetration testing for web applications. This approach combines Generative Adversarial Networks (GANs) with Reinforcement Learning (RL) to enhance the detection of security vulnerabilities in web applications, particularly targeting application-layer attacks such as Cross-Site Scripting (XSS) and SQL Injection. The method utilizes a reinforcement learning approach similar to Sequence Generative Adversarial Nets (SeqGAN), modeling the process of generating attack sequences as a sequential decision-making problem, thereby addressing the limitations of traditional GANs in generating discrete data.

Methodology

The core of the methodology employs Conditional GANs (CGANs), integrating reinforcement learning and semantic tokenization techniques to automate the generation and optimization of attack payloads. The detailed steps are as follows:

Semantic Tagging and Tokenization

- Attack features were first extracted from an XSS attack dataset, and semantic tokenization was used to break down the raw attack samples

into specific syntactic units, such as HTML tags, script parameters, function bodies, hyperlinks, etc.

- Byte Pair Encoding (BPE) was applied to create a semantically structured feature set by reducing redundancy and frequent symbol combinations. These tokens were used to enhance the generator's efficiency and logical consistency in subsequent processes.

Conditional Sequence Generation

- A CGAN model was constructed with a conditional generator and discriminator. The generator guided sequence generation based on conditional attack labels, while the discriminator evaluated the authenticity and attack efficacy of the generated sequences.

- Using policy gradients and Monte Carlo Search from reinforcement learning, the generator iteratively optimized attack sequences, selecting the optimal token combinations for the current state and validating their effectiveness through replay experiments.

GAN Training and Optimization

- The generator and discriminator were initially pre-trained using Maximum Likelihood Estimation (MLE). Subsequently, the generator was trained with conditional labels to generate efficient attack sequences that could bypass Web Application Firewalls (WAFs).

- The discriminator periodically evaluated generated samples and updated its parameters to optimize the overall model performance, ensuring the generated attack sequences could bypass existing WAF rules.

Attack Payload Validation and Model Refinement

- The generated attack payloads were tested on open-source platforms such as ModSecurity WAF and AWS WAF. By evaluating the success and bypass rates, the generator and discriminator's parameters were adjusted to improve the logical structure and diversity of the attack payloads.

Experimental Results

The effectiveness of the proposed method was validated through the following aspects:

GAN Convergence and Generation Performance

- Experimental results showed that the discriminator's loss value rapidly decreased during training and converged after approximately 250 iterations. The generator's loss value also decreased significantly, demonstrating that the model could stably generate highly effective attack samples capable of bypassing firewalls.

Web Application Firewall Bypass Testing

- In ModSecurity WAF tests, the study compared the performance of traditional GANs and the improved CGAN. The results showed that CGAN was more stable in generating realistic attack samples, with a higher proportion of samples successfully bypassing the WAF. In some batches, CGAN achieved a bypass rate as high as 12%.

- In AWS WAF tests, the bypass rate was lower (approximately 8%). However, the study emphasized that even a small number of successful attack samples could pose a significant threat to existing WAF rules and highlighted the potential of the generator to improve WAF signature rules.

Comparison with Existing Studies

- Compared to automated penetration testing methods based on Partially Observable Markov Decision Processes (POMDP) [45], the proposed method demonstrated faster convergence (about one-quarter of the time required by the POMDP approach) and significantly improved adaptability to complex attack patterns through conditionally guided semantic generation strategies.

- Compared to static rule-based detection methods and traditional generation methods, this study showed significant advantages in attack diversity and bypass rates while effectively avoiding mode collapse and long-sequence dependency issues.

4.3.2 AIRL

In 2024, Alexey Sychugov and Mikhail Grekov published a paper titled *"Automated Penetration Testing Based on Adversarial Inverse Reinforcement Learning"* [46]. This paper proposed an automated penetration testing framework based on a technique named Adversarial Inverse Reinforcement Learning (**AIRL**).

Principles of AIRL

- **Inverse Reinforcement Learning (IRL):**
  - The objective is to recover the reward function from expert demonstrations and train an agent to simulate expert strategies.
  - The maximum entropy principle is applied to ensure the generated strategies exhibit diversity.
- **Generative Adversarial Networks (GANs):**
  - Consists of a *Generator* and a *Discriminator*.
  - The Discriminator differentiates between the generated strategies and expert demonstrations, while the Generator maximizes the reward function to produce attack paths similar to expert strategies.

Key Components

- **Generator:** Optimized using policy gradient methods to generate attack strategies. After each iteration, the Generator adjusts its strategy based on feedback from the Discriminator to align more closely with expert behavior.
- **Discriminator:** A binary classification model that takes state-action pairs as input and outputs the probability of the input being from the expert demonstration or the generated strategy. Its training objective is to maximize its ability to distinguish expert demonstrations from generated strategies.
- **Maximum Entropy Principle:** Adds an entropy term to the optimization objective to increase randomness and diversity in the generated strategies, avoiding deterministic solutions. The optimization goal is given by [46]:

$$\pi^* = \text{argmax} \sum_{(s,a)} \pi(a|s) \left[R(s,a) + \lambda H(\pi(\cdot|s)) \right]$$

  where $R(s,a)$ is the reward function, and $H(\pi(\cdot|s))$ is the entropy of the policy.

Expert Data Collection

- **Deep Exploit:** To effectively train the AIRL model, high-quality expert demonstration data is required. The paper utilizes the open-source tool *Deep Exploit* for data collection.
- **Deep Exploit Capabilities:**
  - **Automated Vulnerability Scanning:** Scans network infrastructures to identify potential vulnerabilities.
  - **Exploitation:** Automates the exploitation of identified vulnerabilities to generate complete attack paths.
  - **Data Recording:** Captures state-action pairs during the attack process for training the AIRL model.
- **Content of Expert Data:**
  - Vulnerability types (e.g., CVE identifiers), criticality assessments (threat levels), attack paths, bypass techniques, and system responses (e.g., alerts from intrusion detection systems).
- **Collection Process:** Multiple rounds of testing are conducted using *Deep Exploit*, each focusing on specific network nodes. The collected data is then organized into expert demonstrations to form the AIRL training dataset.

Introduction of Semantic Rewards

Semantic rewards are a key improvement to AIRL, designed to incorporate domain knowledge, enhancing the practicality and training efficiency of the generated strategies.

- **Design of Semantic Rewards:**
  - The reward function incorporates domain-specific semantic signals (e.g., attack effectiveness, security risk assessments) to provide fine-grained guidance to the Generator's output.
  - Formula representation [46]:

    $$r(s,a) = f_{\text{security}}(s,a) + f_{\text{effectiveness}}(s,a) + \text{other\_semantic\_factors}(s,a)$$

    * f_{security}: Evaluates system security based on the attack scenario.

    - $f_{\text{effectiveness}}$: Measures the impact of the attack on the target system.
    - other_semantic_factors: Represents extensible domain-specific metrics.

- **Role of Semantic Rewards:** Semantic rewards impose constraints on the Generator's training process, ensuring the generated strategies not only replicate expert demonstrations but also meet specific domain requirements. These rewards are embedded into the Discriminator, influencing the Generator's gradient updates and stabilizing training outcomes.

Experiments

The paper does not provide specific experimental results. Instead, it proposes six evaluation criteria for assessing AIRL's effectiveness:

1. **Accuracy of Expert Strategy Reproduction:** The ability of the AIRL model to reproduce expert strategies.
2. **Comparison with Real Attack Outcomes:** Measures the alignment between AIRL-generated strategies and actual attack results.
3. **Adversarial Robustness:** The model's resilience against adversarial attacks.
4. **Diversity of Strategies:** Evaluates whether AIRL generates sufficiently diverse strategies.
5. **Training Time and Computational Resources:** Assesses the time and resources required to train the AIRL model.
6. **Robustness to Environmental Changes:** The model's adaptability to changes in information systems, including expert strategies and evolving threats.
7. **Low-Data Training Capability:** The model's ability to learn effectively with limited data.

4.3.3 IAPTS

In 2019, Mohamed C. Ghanem and others published a paper titled *"Reinforcement Learning for Efficient Network Penetration Testing"* [47], which designed an intelligent automated penetration testing system (IAPTS). The following provides a detailed description of the research methods:

Detailed Description of Research Methods

Problem Modeling. The paper first models the penetration testing problem as a complex Partially Observable Markov Decision Process (POMDP). The penetration testing environment is defined as a collection of state space, action space, reward functions, transition probabilities, and observation probabilities.

- **State space:** Includes all machines and their configurations in the network, such as operating systems, open ports, running services, vulnerabilities, network topology, and security settings.
- **Action space:** Covers all penetration testing tasks, such as scanning, vulnerability assessment, privilege escalation, and attack path exploration.
- **Reward function:** Guides the reinforcement learning agent to optimize its behavior, with positive rewards for achieving critical targets or reducing testing time and negative rewards for failures.

Selection and Optimization of Reinforcement Learning Algorithms. The study employs multiple reinforcement learning algorithms to adapt to network testing scenarios of varying scales and complexities, such as the following: **PERSEUS** (Randomized Point-Based Value Iteration), **GIP** (Incremental Pruning Algorithm), and **PEGASUS** (Policy Search Algorithm). Significant optimization was applied to the GIP algorithm by introducing an improved initial belief sampling method and a prioritized experience replay mechanism to enhance efficiency in solving POMDP problems in large network environments. The system dynamically adjusts algorithm selection during different testing stages, such as using policy search algorithms for quickly generating strategies in the early stages and employing incremental pruning for precise optimization in complex later stages.

IAPTS Architecture Design and Functional Modules. The paper presents a modular IAPTS, which can be integrated as an independent module into existing industrial penetration testing frameworks such as Metasploit. The IAPTS architecture includes the following core modules:

- **Reinforcement Learning Module:** Responsible for converting the penetration testing environment into a POMDP model and optimizing testing strategies using reinforcement learning.

- **Knowledge Base and Experience Storage Module:** Records strategies and experiences from past tests to support reuse in future testing tasks.
- **Interaction Module:** Communicates with testing frameworks (e.g., Metasploit) through APIs, translates reinforcement learning outputs (policy graphs) into specific penetration testing tasks, and provides real-time feedback on testing results.
- **Operating Modes:** The system features four operating modes:
    1. Fully Autonomous Mode (performs tasks independently once mature),
    2. Partially Autonomous Mode (executes under expert supervision),
    3. Decision-Making Assistant Mode (provides intelligent suggestions to testers),
    4. Experience Building Mode (captures and stores knowledge by observing human testers).

Data Processing and Optimization Mechanisms. To improve system efficiency, the study introduces detailed designs for input data processing and experience handling:

- **Streamlined Representation of State and Action Spaces:** Records only critical network configurations and tasks related to penetration testing while removing redundant information to reduce computational complexity.
- **Prioritized Experience Replay:** Introduces a mechanism based on past testing experiences, replaying critical strategy sequences to accelerate learning and optimizing strategy applicability based on expert feedback.
- **Reward Mechanism Design:** Initially relies on human experts to provide reward values for guiding reinforcement learning system optimization. Over time, the system learns to automatically generate reward functions to dynamically adjust testing task priorities.

Experiments

Testing Environment and Performance Evaluation. The research constructed test networks of varying scales (from 2 to 100 devices) in a laboratory setting, including typical network topologies such as internet-facing

segments, DMZ areas, internal LANs, and sensitive data storage zones. Performance comparisons between IAPTS, manual penetration testing, and traditional automated tools showed significant advantages of IAPTS in testing time, attack path coverage, accuracy, and resource efficiency. Especially in small and medium-sized networks, IAPTS significantly reduced testing time while discovering complex, non-obvious attack paths often overlooked in manual testing.

IAPTS System Output. The system's final output consists of a series of optimized **policy graphs**, which clearly illustrate the optimal sequence of actions during the penetration testing process. These graphs guide automated tools to efficiently execute testing tasks. Additionally, policy graphs can be stored in the system's knowledge base to support experience reuse in future similar penetration testing scenarios, greatly enhancing the efficiency of regular security audits.

4.3.4 IAPTF

In 2019, Mohamed C. Ghanem's team published the paper titled *"Hierarchical Reinforcement Learning for Efficient and Effective Automated Penetration Testing of Large Networks"* [48], which proposed a system named **IAPTF** (Intelligent Automated Penetration Testing Framework). This framework leverages reinforcement learning techniques, a knowledge management module, and hierarchical modeling methods to provide an intelligent, scalable, and efficient solution for penetration testing in large-scale networks. The paper addresses the issues of high complexity, inefficiency, and repetitive tasks in traditional penetration testing by introducing an intelligent method centered around hierarchical reinforcement learning (HRL). The goal is to overcome the dimensionality challenges posed by network growth through model optimization, knowledge reuse, and task decomposition.

Framework Overview

IAPTF is built on reinforcement learning, modeling penetration testing as a partially observable Markov decision process (POMDP). This approach transforms penetration testing into a sequential decision-making problem optimized through reward mechanisms. By clearly defining state space, action space, reward functions, and transition probabilities, the system captures the complexity and dynamics of the network.

- **State Space:** The system structures detailed information about network devices, including operating systems, open ports, and vulnerability data.
- **Action Space:** Encompasses all key penetration testing tasks such as reconnaissance, vulnerability assessment, privilege escalation, and attack path exploration, ensuring comprehensive task coverage.
- **Reward Function:** Dynamically adjusts based on the success or failure of each test operation, optimizing testing efficiency and reducing resource consumption in the long term.

Hierarchical Reinforcement Learning (HRL)

To address the computational complexity associated with the exponential growth of state spaces in large-scale networks, the paper proposes a **hierarchical reinforcement learning approach based on security clustering**. This method divides the large network into multiple sub-clusters with similar security characteristics and independently models and solves the POMDP for each cluster, significantly reducing computational overhead.

- **First Layer:** Treats each security cluster as an independent POMDP environment.
- **Second Layer:** Models connections between cluster head nodes to address cross-cluster testing tasks, enabling the generation of optimized strategies from local to global levels.

Algorithm Integration and Solver Flexibility

IAPTF incorporates several reinforcement learning algorithms, **PERSEUS** and **GIP**, as well as external solvers like **SolvePOMDP**, to flexibly adapt to various testing scenarios:

- **PERSEUS:** A randomized point-based value iteration algorithm suitable for rapid approximation in large-scale POMDPs, ideal for initial strategy exploration.
- **GIP:** Focuses on generating high-quality policy graphs, appropriate for high-precision scenarios.

This combination balances computational efficiency and result accuracy, while external solvers enhance performance to meet varying network testing demands.

Knowledge Management System

IAPTF integrates a modular knowledge management system to extract, generalize, and store testing experience, supporting long-term knowledge accumulation and reuse:

- **Preprocessing Phase:** Automates the standardization of test results and generates input files for POMDP solvers.
- **Postprocessing Phase:** Utilizes a CLIPS-based expert system to generalize and store policy graphs for future reference, significantly improving efficiency for repetitive tasks and incremental network changes.

Experimental Results

Experiments demonstrate that IAPTF significantly outperforms traditional and manual penetration testing methods in medium and large networks. In networks with 200 devices, it achieves over a **4x improvement in time efficiency** while covering a much greater number of attack paths. By leveraging historical test data to optimize initial beliefs, IAPTF generates higher-quality test results in less time, making it highly suitable for periodic security audits and continuous testing of large-scale networks.

4.3.5 BDI-RL

In 2021, Kexiang Qian and colleagues published a paper titled *"Ontology and Reinforcement Learning Based Intelligent Agent Automatic Penetration Test"* [49], proposed an automated penetration testing framework. The core of the research lies in introducing a Belief-Desire-Intention (BDI) agent and leveraging the advantages of Semantic Web Rule Language (SWRL) and reinforcement learning algorithms to optimize penetration test planning in uncertain and dynamic environments. Below is a detailed analysis of the methodology, presented with long sentences to fully articulate each technical detail.

Ontology-Based Knowledge Modeling

The study constructed an ontology model for penetration testing using the Protégé platform, wherein attack types and relationships between instances were abstracted into multiple semantic relations, such as **hasPermission** (indicating that the attacker has a specific level of access to a target) and **isConnected** (indicating that a connection can be established between targets). Furthermore, the study summarized several typical attack methods based on

the MITRE ATT&CK knowledge base, including information-gathering attacks, configuration attacks, buffer overflow attacks, password attacks, web attacks, sniffing attacks, social engineering attacks, and denial-of-service attacks. This ontology-based modeling provides a formalized representation of knowledge, enabling it to be shared with intelligent systems and laying the foundation for subsequent automated reasoning.

Semantic Reasoning and Rule Engines

The study utilized SWRL rules to extend the OWL ontology model, enabling the expression and reasoning of complex rules. In this part, rules were defined in the structure of preconditions → postconditions and combined with reasoning engines like Hermit to deduce implicit semantic relationships. For instance, one rule in the study stated that if an attacker is connected to a target host and the target has a specific vulnerability (e.g., `MS08-067`), the attacker can exploit that vulnerability to launch an attack. These rules not only uncover implicit relationships but also enhance the agent's understanding of and actions within the target environment.

BDI Agent Architecture Design

The study further extended the traditional BDI agent model by incorporating five core components: agent name (`Ag`), belief set (`B`), desire set (`D`), intention set (`I`), and plan set (`P`). Among these, the belief set represents the agent's cognitive state regarding the environment; the desire set represents the goals the agent aims to achieve; the intention set reflects the actions chosen by the agent to achieve its goals; and the plan set defines the sequences of actions available to the agent. The reasoning and execution mechanisms of the agent are coordinated through trigger events and contextual conditions. For example, when the agent identifies that specific conditions in the target environment are met, it triggers corresponding plans to execute attack actions.

Integration of Reinforcement Learning

The study integrated reinforcement learning with the BDI agent to enhance its adaptability in dynamic and uncertain environments. Specifically, the Q-learning algorithm was adopted, which employs dynamic programming to calculate optimal policies that maximize long-term rewards. At each state, the agent observes the environment, takes actions, and continuously optimizes its policy based on reward feedback. The primary advantage of Q-learning lies in its model-free nature, allowing it to be applied in unknown

or partially observable environments. Additionally, the study compared the SARSA algorithm, a learning method based on actual actions, which further improved the agent's responsiveness to environmental changes.

Experimental Validation and Result Analysis

The experiments were conducted on the Jason simulation platform to simulate penetration testing scenarios and verify the proposed framework's ability to optimize penetration paths in uncertain environments. The experiments utilized a notebook with an i7 processor, and through multiple rounds of simulations, the agent's learning curve was observed. The results demonstrated that the BDI agent, combined with the ontology knowledge base and reinforcement learning, outperformed traditional methods in terms of planning efficiency and decision accuracy.

Chapter 5

Overall Analysis

From the summary of the above papers, We can see that the application of reinforcement learning in the field of network security has achieved rapid development. With the increasing complexity and dynamism of network threats, researchers have begun to explore the application of Deep Reinforcement Learning (DRL) in Automated Penetration Testing (APT). Since 2018, numerous studies have emerged. Traditional research faced challenges such as large action spaces, sparse reward environments, and algorithm adaptability in dynamic network scenarios. Subsequent studies have consistently aimed to address these critical issues. This section will summarize all the aforementioned papers. We first conducted a general analysis of all the studies mentioned above and presented an overall display in a table. Subsequently, we carried out analyses from various aspects.

5.1 Comprehensive Analysis of Research

We conducted a comprehensive analysis of the aforementioned research, as shown in Table 5.1. The table summarizes the names of the various research frameworks and methods, their specific implementation details, and the types of models used, illustrating the diverse explorations undertaken by researchers in applying reinforcement learning to automated penetration testing. This comparative analysis provides valuable insights into the strengths and limitations of each approach.

Table 5.1: Summary of Frameworks and Methods

Framework Name	**Specific Method**	**Model**	**Approach**
NIG-AP (2019)[28]	Based on DQN, it introduces information gain and action cost (calculated by CVSS).	MDP	DQN
AutoPentest-DRL (2020)[2]	It collects real data through Shodan, uses MulVAL to generate attack trees, traverses the attack tree with DFS, simplifies the attack matrix (as input for DQN), calculates reward scores using CVSS, and makes decisions with DQN.	MDP	DQN
ASAP (2020)[3]	It first uses a scanning module to collect information input into MulVAL to generate an attack graph, and then trains with DQN. The reward is based on CVSS.	MDP	DQN
NDSPI-DQN (2021)[4]	It improves DQN with five measures (including Noisy Nets with Gaussian noise, Soft Q-learning, Dueling Architectures, PER, and ICM).	MDP	DQN
CJA-RL (2021)[5]	It generates an attack graph through MulVAL, uses DQN for decision-making, and quantifies vulnerability severity with CVSS.	MDP	DQN
HA-DRL (2021)[6]	It introduces hierarchical agents and action space decomposition. The action space is divided into smaller subspaces, and a dedicated reinforcement learning agent is assigned to each subspace.	MDP	DQN
CRLA (2022)[31]	It uses a hierarchical agent architecture, with a core identical to HA-DRL. It employs Dueling DQN and introduces the QMIX mixing network mechanism to aggregate the Q-values of each agent to guide decision-making.	MDP	DQN
SmartGrid-PTDRL (2022)[7]	It designs three actions: "stop," "record," and "replay." It causes voltage instability in the power grid through network attacks to train the agent to learn the best timing and sequence of attacks in different operating environments.	MDP	DQN
ND3RQN (2022)[8]	It introduces LSTM to capture long-term historical information, combines Double DQN and Dueling DQN architectures to reduce Q-value overestimation and improve the accuracy of value estimation. It also introduces Noisy Nets with random noise similar to NSDPI-DQN.	POMDP	DQN

Framework Name	Specific Method	Model	Approach
Improved-PenBox (2022)[32]	It adds PenBox to NAS (the authors only selected the 6 most commonly used tools and modes, while the original functionality includes 52 tools and 363 modes), and then uses the interface to call the DQN in NAS for training.	MDP	DQN
OAKOC (2022)[33]	It proposes reward adjustment and state adjustment, and based on the OAKOC framework, integrates network terrain into the attack graph and models firewalls as obstacles.	MDP	DQN
HDRL (2023)[34]	It introduces "expert experience," uses a hierarchical architecture, and breaks down penetration testing into several subtasks, which are handled by multiple specialized agents.	MDP	DQN
DQfD-AIPT (2023)[9]	It also introduces "expert experience" and uses PER and N-step return mechanisms. Pre-training uses "supervised loss" to guide the agent to imitate expert behavior.	MDP	DQN
MDDQN (2023)[35]	It uses MulVAL to generate an attack graph, traverses the attack graph with DFS to form a transition matrix, and uses DDQN with epsilon-greedy strategy and experience replay.	MDP	DQN
INNES (2023)[36]	It proposes DQN_valid, eliminating the dependence on "expert experience" and other "prior knowledge." The core is to set a known node set K and a valid action set V, and dynamically expand V to gradually build the network's state representation during exploration.	MDP	DQN
HER-PT (2024)[10]	It introduces HER (Hindsight Experience Replay), converting failed experiences into positive rewards, enabling the model to learn from failures and improving learning efficiency.	MDP	DQN
DynPen (2024)[19]	It uses "expert experience" as an auxiliary, has an HDRL (Hierarchical Deep Reinforcement Learning) structure, and adds a trajectory recording module, monitoring module, and backtracking module.	MDP	DQN

Framework Name	Specific Method	Model	Approach
DRLRM-PT (2024)[11]	It designs a reward machine, which is essentially a state machine, breaking down penetration testing into a series of manageable subtasks and increasing interpretability and transparency. The reward machine is used to decompose tasks, generate subtask sequences, and assign independent reward functions to each subtask.	POMDP	DQN
DL-IAPTS (2024)[37]	It proposes a dynamic matrix representation of the current state of network intrusion, continuously updating the threat matrix through MulVAL and Shodan for DQN training and updating.	POMDP	DQN
MLAE-WA (2020)[41]	It uses three-layer embedding (action feature embedding, network structure embedding, service vulnerability embedding) to help the agent better understand and select actions, and combines it with the Wolpertinger Architecture (WA).	MDP	AC
DAA (2021)[42]	It proposes the Double Agent Architecture, designing a separate training environment for it based on the NAS environment. The structure agent is responsible for exploring and identifying the network architecture, and the exploit agent is responsible for selecting appropriate services to attack on the chosen host.	MDP	AC
HAE (2022)[39]	It introduces MITRE ATT&CK on the basis of MLAE-WA, effectively managing and simplifying the action space.	MDP	AC
MAR-WA (2023)[12]	It more finely decomposes actions into three parts, enriches attack technique modeling based on MITRE ATT&CK, introduces Epsilon-Wolpertinger, optimizes training efficiency, and reduces computational complexity.	MDP	AC
CLAP (2022)[13]	It uses the PPO algorithm combined with Generalized Advantage Estimator (GAE), introduces Random Network Distillation (RND) to provide curiosity rewards, introduces weighted Chebyshev decomposition critic to convert multi-objective reward vectors into a single scalar, and proposes the Coverage Masking Mechanism to encourage exploration and optimize learning efficiency.	MOMDP	PPO

Framework Name	Specific Method	Model	Approach
EPPTA (2023)[14]	It uses PPO combined with an implicit belief module, storing historical data through LSTM. It integrates EPPTA into the Sample Factory framework, significantly reducing data transmission overhead through shared memory mechanisms and double buffering sampling strategies to accelerate convergence speed.	POMDP	PPO
RLAPT (2023)[15]	It uses PPO, combined with action space decomposition. It introduces an action mask to filter out invalid actions under the current state, and a probability mask to exclude attack parameters irrelevant to the current attack selection. The reward function directly assigns scores, such as rewarding 5 points for successfully discovering a new node.	POMDP	PPO
SeqGAN-PT (2023)[44]	It uses a Generative Adversarial Network (GAN) with Conditional Sequence Generation, composed of a Generator and a Discriminator. It preprocesses attack samples through Semantic Tokenization and BPE to reduce the feature space dimension. It uses GAN to generate sample sequences with specific attack labels, then optimizes the generation strategy with Policy Gradient, and predicts the expected rewards of attack sequences in different states through Monte Carlo Search to enhance the effectiveness and diversity of generated samples.	GANs	Other
AIRL (2024)[46]	It proposes AIRL (Adversarial Inverse Reinforcement Learning), combining Semantic Rewards and expert data collection. AIRL is a method that combines GANs and IRL. Semantic rewards are embedded in the discriminator function, guiding the generator through the reward function to produce strategies that better fit real attack scenarios.	GANs	Other

Framework Name	Specific Method	Model	Approach
IAPTS (2019)[47]	It uses a combination of PERSEUS, GIP, and PEGASUS, adopting a flexible multi-algorithm solution strategy according to different situations to balance accuracy and computational efficiency under limited resources. In the early stages of the system, the reward mechanism is optimized through human expert feedback, and after the system matures, it can independently evaluate rewards and continuously optimize strategies based on past experience.	POMDP	Other
IAPTF (2023)[48]	It also uses PERSEUS and GIP. It proposes Security-Based Clustering, dividing large-scale networks into multiple security clusters, each treated as a small network for independent processing. Then, it establishes a separate POMDP model for each cluster, and subsequently creates a POMDP model based on the "head nodes" (most vulnerable nodes) of each cluster, effectively reducing computational complexity.	POMDP	Other
BDI-RL (2021)[49]	The core is to use Belief-Desire-Intention (BDI) agents, combined with Semantic Web Rule Language (SWRL). It establishes an ontology model for penetration testing using the Protégé platform. It extends the OWL ontology model with SWRL rules to achieve the expression and reasoning capabilities of complex rules. It combines BDI agents with Q-learning to enhance their adaptability in dynamic and uncertain environments.	MDP	Other

5.2 Historical Development and Key Contributions

5.2.1 Foundational Stage (2018–2020)

Early research on Deep Reinforcement Learning (DRL) opened new pathways for automated penetration testing. For instance, **AutoPentest-DRL** (2020) [2] introduced a DRL-based automated penetration testing framework, which utilized Shodan to collect real-world vulnerability data and combined it with MulVAL to generate attack graphs, validating the feasibility of automated penetration testing. While this framework comprehensively covered processes from data preprocessing to attack path generation, it lacked scalability for large networks and adaptability to dynamic environments. Additionally, **NIG-AP** (2019) [28] proposed an attack path selection method aimed at reducing information entropy, providing effective guidance strategies in sparse reward settings.

Another milestone was **ASAP** (2020) [3], which achieved significant efficiency improvements in complex network scenarios by optimizing the reward function using CVSS scores. However, its reliance on static attack graph generation limited its adaptability to dynamic environments. These studies modeled penetration testing problems as Markov Decision Processes (MDPs), laying the groundwork for subsequent research.

5.2.2 Domain Expansion (2021–2023)

Starting in 2021, the research emphasis shifted from infrastructure optimization to exploring more complex models and mechanisms. Hierarchical agent models and experience replay mechanisms became hot topics during this period. For example, **HA-DRL** (2021) [6] introduced a hierarchical agent architecture, improving learning stability and efficiency through action space decomposition. Meanwhile, **NDSPI-DQN** (2021) [4] integrated multiple DQN extension techniques, such as prioritized experience replay and intrinsic curiosity models, demonstrating outstanding performance in addressing sparse reward issues and large action spaces. Furthermore, studies based on MulVAL tools, such as **CJA-RL** (2021) [5] and **OAKOC** (2022) [33], incorporated network topology information into reinforcement learning models, enhancing scenario adaptability.

During this period, research on Proximal Policy Optimization (PPO) algorithms also gained traction. Representative works include **CLAP** (2022) [13], **EPPTA** (2023) [14], and **RLAPT** (2023) [15]. Notably, CLAP proposed a lightweight policy optimization framework that accelerated conver-

gence by adjusting update frequency, EPPTA improved reward mechanisms to better adapt to dynamic environments, and RLAPT balanced strategy stealthiness and effectiveness through multi-objective optimization.

Parallel to this, actor-critic (AC) algorithms achieved significant advancements. **DAA** (2021) [42] introduced a Double Agent Architecture that combined global scheduling with local task decomposition. **HAE** (2022) [39] enhanced adaptability to dynamic environments by incorporating a state prediction module within the AC framework. **MAR-WA** (2023) [12] improved policy optimization in distributed network environments through multi-agent collaboration and attention mechanisms. Notably, these AC algorithm advancements were primarily led by the Hoang Viet Nguyen team, which has focused on applying AC algorithms to penetration testing since **MLAE-WA** (2020) [41] by continually optimizing hierarchical structures and reward mechanisms.

Domain applications deepened during this period, as exemplified by the approach named **SmartGrid-PTDRL** (2022) [7], which targeted intelligent grid scenarios.

In 2023, significant progress was made in addressing expert knowledge and dynamic action space constraints. For example, **DQfD-AIPT** (2023) [9] accelerated learning through supervised pretraining, though its reliance on expert knowledge limited scalability. In contrast, **INNES** (2023) [36] dynamically constructed effective action spaces, significantly reducing invalid exploration and further enhancing model performance. This stage of research showcased a flourishing diversity of approaches.

5.2.3 Emerging Trends (After 2023)

Recent research focuses on improving dynamic adaptability and environmental responsiveness. **HER-PT** (2024) [10] was the first to introduce hindsight experience replay (HER) into penetration testing, significantly enhancing learning efficiency in sparse reward scenarios through the reconstruction of failed experiences. **DynPen** (2024) [19] combined hierarchical architectures, monitoring modules, and replay mechanisms, demonstrating exceptional adaptability in rapidly changing network environments.

Additionally, Generative Adversarial Network (GAN) technology has been increasingly applied to optimize attack path generation. For instance, **AIRL** (2024) [46] and **SeqGAN-PT** (2023) [44] leveraged adversarial learning to optimize strategies, overcoming computational efficiency bottlenecks in traditional MDP models and opening new research directions for penetration testing.

5.3 Comparative Analysis

5.3.1 Model Framework: MDP vs. POMDP

In the early stages of automated penetration testing research, Markov Decision Processes (MDPs) were the mainstream choice due to their assumption of fully observable states. For example, **AutoPentest-DRL** (2020) [2] and **NDSPI-DQN** (2021) [4] simplified the description of network states to quickly plan optimal attack paths. However, as the complexity and dynamics of real-world network environments became apparent, Partially Observable Markov Decision Processes (POMDPs) gained prominence. POMDPs align better with practical penetration testing scenarios. Models such as **ND3RQN** (2022) [8], **DRLRM-PT** (2024) [11], and **DL-IAPTS** (2024) [37] incorporated LSTMs, reward machines, and dynamic threat matrices, enabling agents to make more precise decisions under incomplete information.

However, the transition from MDPs to POMDPs is not straightforward. The introduction of POMDPs increases the complexity of algorithm design, requiring mechanisms such as memory structures (e.g., LSTMs) to capture historical information or the integration of domain knowledge (e.g., MITRE ATT&CK knowledge base) to optimize reward design and improve learning efficiency. Moreover, strategy generation under partially observable states in POMDP models demands greater robustness and adaptability.

5.3.2 Action Space Optimization

As network scales grow, the explosive increase in action space becomes a major challenge for Deep Reinforcement Learning (DRL) in penetration testing. Hierarchical methods and dynamic action constraint techniques provide effective solutions to this problem. Hierarchical methods, such as **HA-DRL** (2021) [6] and **DynPen (2024) [19]**, decompose actions into global and local subtasks, segmenting the action space into manageable subsets. For instance, **DynPen** (2024) [19] introduces an upper-level agent to handle global task scheduling, while lower-level agents manage subtasks such as vulnerability exploitation and credential connection, significantly reducing action complexity.

On the other hand, dynamic action constraint techniques, such as **INNES** (2023) [36] and **DQN_valid**, dynamically adjust action sets to further optimize resource allocation. **INNES** (2023) [36] proposed a method for constructing effective action subsets, reducing meaningless exploration and enhancing applicability in large-scale scenarios. However, these methods often

require additional modules (e.g., monitoring and replay modules in **DynPen** (2024) [19]) or rely on prior knowledge bases (e.g., expert knowledge bases), which improve efficiency at the cost of increased system complexity.

5.3.3 Reward Mechanism

The sparse reward problem has been a bottleneck for reinforcement learning in penetration testing, especially in dynamic and complex network environments. This issue has persisted since early studies like **NAS** (2018) [1] and **AutoPentest-DRL** (2020) [2]. Traditional approaches typically designed rewards based on final goals or single rewards, such as node exploitation success. However, such simplistic mechanisms struggle in complex environments. To address this, many studies have optimized reward mechanisms through hindsight experience replay and task decomposition. For instance, **HER-PT** (2024) [10] introduced a "goal resetting" method that reconstructs failed experiences into positive rewards, increasing the proportion of positive samples and addressing the sparse reward problem. **DRLRM-PT** (2024) [11] utilized reward machines to decompose tasks, designing independent reward functions for each subtask to enhance training efficiency and improve the interpretability of generated strategies.

Moreover, improvements to reward mechanisms based on MulVAL attack graphs have drawn attention. For example, **ASAP** (2020) [3] embedded CVSS scores into attack graph nodes to optimize reward functions, while **MDDQN** (2023) [35] leveraged an extended state transition matrix to alleviate the sparse reward issue significantly.

5.3.4 Integration of Expert Knowledge and Experience-Driven Methods

Expert knowledge and experience in replay techniques have been widely applied in reinforcement learning for penetration testing. For example, **DQfD-AIPT** (2023) [9] and **HER-PT** (2024) [10] leveraged expert knowledge to rapidly improve initial policy efficiency. However, overreliance on expert knowledge may limit the model's adaptability to diverse scenarios. To mitigate this, **INNES** (2023) [36] proposed reducing dependency on prior knowledge by dynamically constructing state representations, while also combining dynamic action constraint techniques to enhance model robustness.

It is worth noting that the integration of hierarchical agents with expert knowledge is becoming a trend. For example, **DynPen** (2024) [19] utilized a Hierarchical Deep Reinforcement Learning (HDRL) architecture

to decompose tasks and incorporated expert knowledge bases for guidance, significantly improving learning efficiency and the reliability of strategy generation.

5.3.5 Deep Integration of MulVAL Attack Graphs and Reinforcement Learning

Attack graphs generated by MulVAL provide critical prior knowledge for penetration testing, but their deep integration with reinforcement learning models remains a key challenge. For instance, **ASAP** (2020) [3] optimized reward functions using CVSS scores embedded in attack graph nodes, while **MDDQN** (2023) [35] extended state transition matrices to enhance path selection accuracy. Additionally, **DL-IAPTS** (2024) [37] employed dynamic threat matrices, combining MulVAL and Shodan tools to generate real-time updated attack graphs, greatly enhancing the model's adaptability to dynamic networks. **DRLRM-PT** (2024) [11] went further by decomposing tasks through reward machines, embedding attack graph information into subtask sequence reward design, achieving efficient and interpretable strategy generation.

5.3.6 Open Source Platforms and Real-World Validation

Although simulation environments provide standardized testing for automated penetration testing, many studies have attempted to combine real-world scenarios for validation. For example, **HER-PT** (2024) [10] validated its performance in dynamic network scenarios on the NAS platform and further tested its transferability in real network environments. **Improved-PenBox** (2022) [32] simulated real attacks in the network environment of the European Space Agency (ESA), demonstrating the adaptability and feasibility of its model. Additionally, **DRLRM-PT** (2024) [11] verified the effectiveness of reward machines through CyberBattleSim and tested the impact of different reward designs on performance.

By combining simulation and real-world data, such as **DL-IAPTS** (2024) [37], which leveraged Shodan to collect real-world network information and incorporated it into simulation environments to construct semi-realistic test scenarios, this approach further enhances the credibility of experimental results.

5.4 Key Challenges and Future Directions

5.4.1 Dynamic Adaptability

Current models like **DynPen** (2024) [19] demonstrate certain advantages in dynamic network environments, but they suffer from high resource consumption and computational complexity, particularly when dealing with frequently changing network topologies. The real-time responsiveness of such models remains insufficient. Future research could focus on developing lighter-weight dynamic frameworks, such as integrating event-triggered mechanisms with deep reinforcement learning-based dynamic strategies to reduce resource consumption and optimize response speed. Additionally, multi-level dynamic scheduling strategies can be explored, leveraging intelligent task decomposition and distributed processing to improve efficiency.

5.4.2 Knowledge Dependency and Autonomy

Current methods partially rely on expert knowledge, such as predefined vulnerability databases. For example, **DQfD-AIPT** (2023) [9], **DynPen** (2024) [19], and **AIRL** (2024) [46] have all applied "expert experience". However, overreliance on these external knowledge sources may limit the scalability and autonomy of models. Future approaches could integrate generative AI and adaptive learning technologies to dynamically generate attack graphs or infer unknown vulnerability states, gradually reducing dependency on fixed knowledge bases. Enhancing agents' ability to learn in unsupervised environments and utilizing knowledge transfer techniques for cross-scenario adaptation are promising directions.

5.4.3 Scalability in Large-Scale Networks

Existing methods face computational bottlenecks and challenges in distributed learning when addressing large-scale network scenarios. For instance, in complex networks with thousands of nodes, traditional hierarchical modeling methods struggle with parallel processing and dynamic load balancing. For example, **NDSPI-DQN** (2021) [4], **HA-DRL** (2021) [6], and **CRLA** (2022) [31] all exhibit this issue. Future research could explore the combination of Graph Neural Networks (GNNs) and reinforcement learning to handle complex network structures. Moreover, leveraging cloud computing and edge computing technologies may enable distributed parallel learning in large-scale networks.

5.4.4 Generative Models and Adversarial Learning

Generative Adversarial Network (GAN) technologies, such as **SeqGAN-PT** [44] and **AIRL** [46] have shown initial potential, but their application to multi-agent collaborative learning and adversarial environments remains underdeveloped. Future research could further explore GAN applications in complex attack scenarios, such as adversarial training, dynamic honeypot defense generation, and multi-agent cooperative strategy optimization. These advancements could redefine the field of automated network security, particularly in proactive defense and adaptive security applications.

5.4.5 Comprehensive Benchmarking

Building standardized and comprehensive testing benchmarks is essential to advance research. Although experimental platforms such as **NAS** (2019) [1] and Microsoft's **CyberBattleSim** (2020) [25] already exist, future benchmarking platforms should include the following based on specific needs:

- Dynamically evolving real-world network topologies with high-frequency changes in nodes and connections.
- Diverse threat scenarios, such as zero-day vulnerability attacks and multi-stage attack chains.
- Dynamic attack-defense games incorporating honeypot technology, proactive defense, and real-time monitoring.

Openly shared simulation platforms and standardized datasets could not only improve comparability in research but also accelerate the practical application of new technologies.

5.4.6 Explainability and Security

As reinforcement learning algorithms are increasingly applied in penetration testing, their explainability and the credibility of results become critical challenges. For instance, current DRL models lack intuitive explanation mechanisms when generating attack strategies, making it difficult for network administrators to understand optimization decisions. The reward machine proposed in **DRLRM-PT** (2024) [11] provides us with a new solution direction. Future research could incorporate tools like reward machines to introduce rule-based model interpretation frameworks, enhancing robustness against adversarial environments and ensuring greater reliability and security in real-world applications.

5.4.7 Interdisciplinary Integration

With the growing complexity of network environments, no single technology can address all challenges. Future research directions could incorporate interdisciplinary knowledge, such as game theory from economics to optimize attack-defense strategies, sociology, and behavioral science to simulate attacker behavior, or IoT and industrial control system (ICS) data to enhance agents' adaptability to real-world scenarios. For example, **SmartGrid-PTDRL** (2022) [7] focuses on cyber-physical scenarios, implying that using IoT or industrial control system data can improve an agent's adaptability to real-world environments.

5.5 Discussion

Reflecting on research progress from 2018 to 2024, significant advancements have been made in addressing the complexity and dynamism of network threats. From early experimental frameworks based on Markov Decision Processes (MDPs) to the introduction of Partially Observable Markov Decision Processes (POMDPs), researchers have tackled challenges like action space explosion, sparse rewards, and dynamic network adaptability through innovative algorithm architectures, dynamic adaptation mechanisms, and domain knowledge integration. These advancements have not only achieved theoretical breakthroughs but also demonstrated exceptional practical potential in diverse application scenarios.

Looking forward, integrating adversarial learning techniques with GANs and reinforcement learning, leveraging GNNs for large-scale network optimization are expected to reshape the research landscape of automated penetration testing. Furthermore, optimizing real-time response mechanisms in dynamic environments, deep interdisciplinary knowledge integration, and establishing standardized testing benchmarks will enhance scalability and practical application value in this field. This research direction will not only continue to drive advancements in cybersecurity technologies but also provide robust support for building safer and smarter future networks.

Chapter 6

Experimental Evaluation

6.1 Experimental Environment

All experiments in this study were conducted on a MacBook Air equipped with a 1.6 GHz dual-core Intel Core i5 processor and 16 GB of DDR3 RAM. The experimental platform utilized was **NAS** (2019) [1], a simulation environment specifically designed for cybersecurity research. NAS was chosen because it provides standardized network environments and attack scenarios, ensuring the reproducibility and comparability of experimental results.

6.2 Research Objective

This study aims to conduct a comprehensive evaluation of the effectiveness of automated penetration testing through a systematic comparison and analysis of various reinforcement learning (RL) algorithms. By empirically assessing the performance of different RL agents in diverse penetration testing scenarios, this research seeks to establish a set of quantitative metrics and methodological guidelines for evaluating the efficiency and reliability of automated penetration testing approaches.

6.3 Experimental scenarios

There are 9 benchmark scenarios in NAS. Their descriptions are shown in Table 6.1.

Scenario	Subnets	Hosts	OS Count	Service Count	Process Count	Exploit Count	Privilege Escalation Count	Description
tiny	3	3	1	1	1	1	1	Minimal configuration with a single OS, service, process, and exploit
tiny-small	4	5	2	3	2	3	2	A configuration between tiny and small
tiny-hard	3	3	2	3	2	3	2	A more challenging tiny configuration
small	4	8	2	3	2	3	2	Standard public network configuration
small-linear	6	8	2	3	2	3	2	Linear topology; the two middle subnets are not directly connected
small-honeypot	4	8	2	3	2	3	2	Contains a honeypot host (located at subnet 3,2)
medium	5	16	2	5	3	5	3	Public subnet configuration
medium-single-site	1	16	2	5	3	5	3	A single subnet with a vulnerable host
medium-multi-site	6	16	2	5	3	5	3	Main site with multiple remote sites across different subnets

Table 6.1: Descriptions of benchmark scenarios

6.4 Experiment 1: Policy Performance of QL, DQN, and QLReplay Agents

6.4.1 Script Implementation

Training and policy execution scripts were developed for the QL Agent and QL Replay Agent, based on the existing implementation of the DQN (Deep Q-Network) agent.

6.4.2 Training Process

Across all benchmark scenarios provided by NAS:

- Default parameter settings were used.
- Each agent was trained for 7,000 episodes per scenario.
- The best-performing policy model during training was saved.

Rationale for Selection:

- Training for 7,000 episodes ensures sufficient learning and convergence.
- Saving the best-performing policy reflects the optimal performance of the algorithms.

6.4.3 Testing and Evaluation

To assess the stability and effectiveness of the trained policies:

- Each saved optimal policy was tested for 100 episodes.
- Key performance metrics were collected:
  - **Attack Success Rate:** Evaluates the reliability of the policy.
  - **Steps** Measures the efficiency of the policy.

6.4.4 Experimental Results and Analysis

Attack Efficiency Evaluation

From the attack step data (Table 6.2), the following key findings can be drawn:

- The QL algorithm performs best in the *tiny-small* scenario (steps: 9.89), while DQN achieves the lowest step count in the *tiny-hard* scenario (steps: 7.16), but fails to complete a large number of scenarios (marked as N/A), indicating insufficient generalization ability.
- QLReplay demonstrates advantages in complex scenarios. For example, in the *medium-multi-site* scenario, it completes the attack in 18.77 steps, significantly outperforming random attacks (1023.19) and brute-force methods (426.32).
- Among baseline algorithms, brute-force performs relatively best in the *small-linear* scenario (steps: 249.28), but its overall efficiency remains 1-2 orders of magnitude lower than QLReplay.
- Random attacks perform the worst across all scenarios, validating the necessity of optimizing attack strategies using intelligent algorithms.

Table 6.2: Average attack steps for different agents across all benchmark scenarios

Agent	Benchmark Scenarios								
	Tiny	Tiny-Small	Tiny-Hard	Small	Small-Linear	Small-Honeypot	Medium	Medium-Single-Site	Medium-Multi-Site
DQN	7.16	N/A	6.18	N/A	N/A	N/A	N/A	N/A	N/A
QL	9.89	17.05	6.96	106.68(81/100)[a]	29.28	20.67	26.59	12.45	45.83
QLReplay	7.82	11.7	7.14	19.18	20.7	14.19	30.15	164.73(44/100)[a]	18.77
Bruteforce	61.4	176	73.22	249.28	346.73	232	773.08	210.48	426.32
Random	110.53	299.63	149.37	473.06	526.91	481.09	1332.97	636.31	1023.19

[a] Values in parentheses indicate success count/total attempts (e.g., 106.68 (81/100)).
"N/A" denotes scenarios where the algorithm failed to complete training.
All values represent attack steps (lower is better); training performed over 7000 episodes.

Training Time Efficiency

From the training time data (Table 6.3), the following observations can be made:

- The QL algorithm has a significant advantage in terms of time efficiency. For example, in the *medium* scenario, its training time is 11,788.94 seconds, only 7.5% of QLReplay's time (157,456.7 seconds).
- QLReplay takes up to 666,677.48 seconds in the *medium-multi-site* scenario, indicating that its computational complexity increases exponentially with scenario complexity.
- DQN achieves relatively short training times in simple scenarios (e.g., *tiny*, 804.99 seconds), but fails to complete training in complex scenarios (marked as N/A), suggesting that its network structure may not be well-suited for complex penetration testing tasks.

Table 6.3: Training time for different agents across all the benchmark scenarios

Agent	Benchmark Scenarios								
	Tiny	Tiny-Small	Tiny-Hard	Small	Small-Linear	Small-Honeypot	Medium	Medium-Single-Site	Medium-Multi-Site
DQN	804.99	N/A	359.77	N/A	N/A	N/A	N/A	N/A	N/A
QL	75.71	196.96	64.57	494.57	692.48	555.59	11788.94	7907.85	8002.37
QLReplay	1197.32	3364.86	1201.3	32323.14	42077.57	24402.87	157456.7	184575.22	666677.48

"N/A" denotes scenarios where the algorithm failed to complete training.
All values represent training time in seconds; training performed over 7000 episodes.

Comprehensive Performance Evaluation

Based on Pareto frontier analysis, the following conclusions can be drawn:

- For time-sensitive tasks, QL is the optimal choice. In the *small-honeypot* scenario, it achieves an optimal balance between attack steps (29.28) and training time (555.59 seconds).
- For tasks prioritizing success rate, QLReplay is preferable. In the *medium-single-site* scenario, it completes 44 out of 100 attacks in 164.73 steps. Although it takes a long time (184,575.22 seconds), its success rate is significantly higher than that of random algorithms (0/100), and it achieves a 100% success rate in other scenarios.
- DQN performs reasonably well in simple scenarios, but its failure rate (N/A marks accounting for 85.7%) suggests the need for improvements in network structure design.

Validation of Method Effectiveness

The experimental results validate the core hypotheses of this study:

- **Effectiveness of the quantitative evaluation system**: The dual-dimension evaluation (steps-time) effectively distinguishes algorithm characteristics (QL prioritizes efficiency, QLReplay prioritizes effectiveness).
- **Differentiability of benchmark scenarios**: The step standard deviation across the nine scenario types reaches 382.5 (for random algorithms), demonstrating the complexity of the test set.
- **Practicality of the framework**: In the *medium-multi-site* scenario, QLReplay achieves an attack step count of 18.77, only 4.4% of brute-force attacks, proving the potential of reinforcement learning in automated penetration testing.

6.5 Experiment 2: Cross-Benchmark Scenario Performance Evaluation

To evaluate the scalability and robustness of the agents, additional experiments were conducted in three benchmark scenarios: **Tiny**, **Small**, and **Medium**.

6.5.1 Experimental Setup

- For each benchmark scenario:
  - **Tiny:** 1,500 episodes
  - **Small:** 3,500 episodes
  - **Medium:** 8,000 episodes
- Each agent (QL, QLReplay, DQN) was run 10 times with default parameters and random seeds.
- The return values from each run were averaged, and episode-return curves were plotted to compare agent performance and convergence speed.
- Random Agent and Brute Force Agent were tested, each for 10 runs. Their average results were used as baselines, marked as horizontal lines in the curves.

6.5.2 Analysis Metrics

- **Convergence Speed:** Measured by comparing the speed at which each agent achieved stable performance with increasing episodes.
- **Baseline Comparison:** Performance thresholds referenced from Random Agent and Brute Force Agent results.
- **Total Training Time:** Indicates the trade-off between algorithm efficiency and practical applicability.

6.5.3 Experimental Results

Episode-Return Performance

Figures 6.1, 6.2 and 6.3 are for the **Tiny**, **Small**, and **Medium** benchmark scenarios, and are used to illustrate the comparative performance of QL, QLReplay, and DQN agents. The key observations are as follows:

- **Tiny Scenario:** QLReplay demonstrates a significantly faster convergence compared to QL, achieving near-optimal returns within the first 300 episodes. In contrast, DQN exhibits slightly slower convergence, stabilizing after 400 episodes. Despite initial fluctuations, QLReplay consistently maintains higher returns.

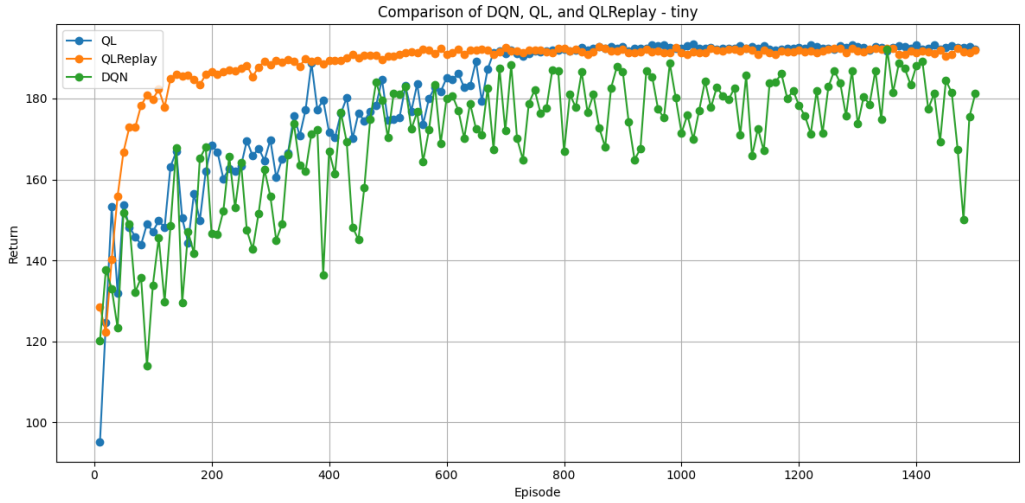


Figure 6.1: Comparison of QL, QLReplay, and DQN in tiny scenario.

- **Small Scenario:** The performance gap between QL and QLReplay becomes more apparent in this scenario. QLReplay reaches convergence earlier and stabilizes at higher returns compared to QL. Notably, DQN does not produce usable results in this scenario due to excessive computational requirements.
- **Medium Scenario:** QLReplay maintains its superior performance in larger-scale environments, achieving stable returns significantly faster than QL. DQN remains infeasible for this scale, highlighting the computational limitations of the approach.

Total Training Time

The average training time (in seconds) for each agent across 10 runs is summarized in Table 6.4. Key observations are as follows:

- **Tiny Scenario:** QLReplay requires roughly 10x the training time of QL but delivers substantially improved performance and faster convergence.
- **Small Scenario:** The gap in training time widens, with QLReplay requiring approximately 25x the time of QL. However, the faster convergence and higher stability may justify the trade-off in larger environments.

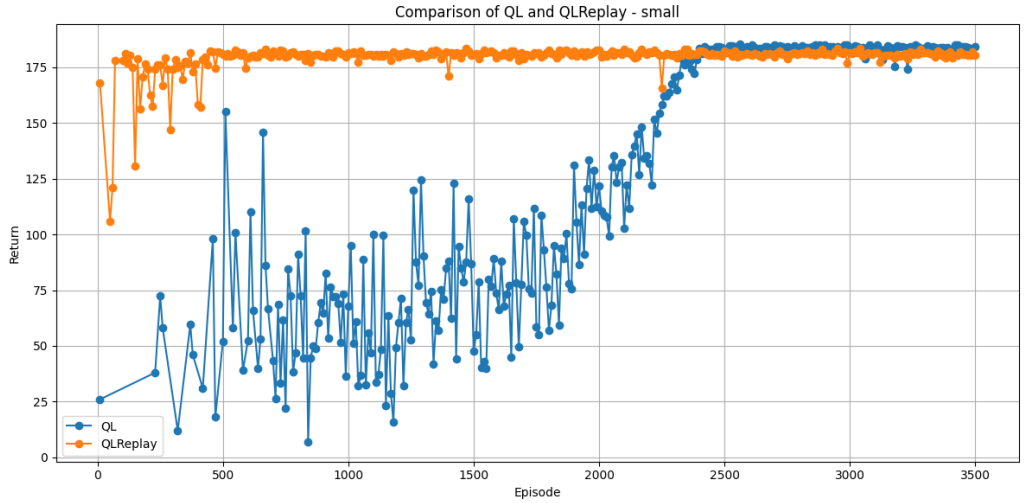


Figure 6.2: Comparison of QL and QL-Replay in small scenario.

- **Medium Scenario:** The computational cost of QLReplay escalates dramatically, requiring over 83,000 seconds on average. However, QL remains computationally affordable while still achieving acceptable performance.

Table 6.4: Average total training time for DQN, QL and QLReplay across the three test scenarios

Agent	**Tiny**	**Small**	**Medium**
DQN	861.44 s	N/A	N/A
QL	13.76 s	330.07 s	1908.92 s
QL-Replay	135.79 s	8289.30 s	83817.82 s

Discussion

- **Trade-Off Analysis:** While QLReplay demonstrates superior performance in return and convergence speed, the trade-off is its significantly higher computational cost. This makes it suitable for applications where performance is critical and computational resources are ample.
- **Scalability:** QLReplay showcases robustness across all scenarios, including the large-scale **Medium** environment. In contrast, DQN is

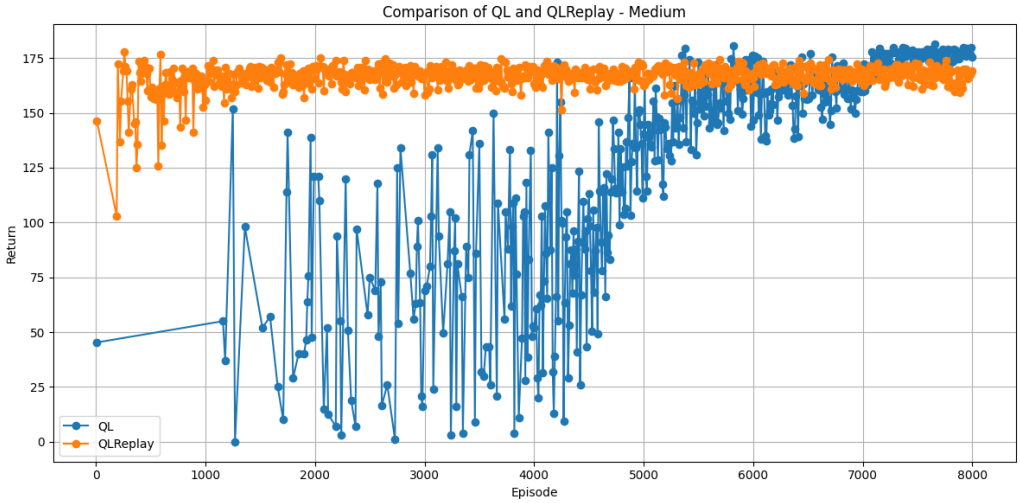


Figure 6.3: Comparison of QL and QL-Replay in medium scenario.

limited by its computational demands, and QL struggles to maintain competitive returns in larger environments.

- **Practical Implications:** The results highlight the importance of balancing computational cost and performance requirements when selecting an agent for real-world applications.

Chapter 7

Conclusion

This research report has provided a comprehensive and detailed overview of the application of reinforcement learning in automated penetration testing. Through a systematic literature review and analysis of over 30 relevant papers, we have identified key contributions, methodologies, and advancements in this field. Our analysis reveals that while reinforcement learning has shown great promise in improving the efficiency and effectiveness of penetration testing, several critical challenges remain. These challenges include the need for more scalable algorithms capable of handling large and complex network environments, the issue of sparse rewards which can hinder effective learning, and the need for greater adaptability to dynamic network changes.

- **Scalability**: Current algorithms often struggle with large-scale networks due to the exponential growth of state and action spaces. Future research should focus on developing more efficient and scalable algorithms, such as those leveraging Graph Neural Networks (GNNs) to handle complex network structures.
- **Sparse Rewards**: The issue of sparse rewards remains a significant challenge, as it can hinder effective learning in reinforcement learning models. Future work should explore methods to address this issue, such as incorporating intrinsic motivation or designing more sophisticated reward mechanisms.
- **Dynamic Adaptability**: Networks are dynamic and constantly changing, requiring algorithms to adapt quickly to new conditions. Future research should aim to develop algorithms that can dynamically adjust to changes in network topology and security policies.

In addition, a series of experiments were conducted with QL, QLReplay, and DQN agents across nine benchmark scenarios provided by NAS. The ex-

perimental results further validate the effectiveness of different reinforcement learning algorithms in various penetration testing scenarios. We have demonstrated that while some algorithms perform well in small-scale networks, they face significant difficulties in larger networks due to the rapid growth of state and action spaces. This highlights the need for future research to focus on developing more efficient and scalable algorithms. Additionally, our experiments have shown the importance of balancing computational cost and performance requirements when selecting an agent for real-world applications.

- **Algorithm Performance**: QLReplay demonstrated superior performance in complex scenarios, achieving significantly fewer attack steps compared to other algorithms. However, it requires much longer training times, making it less suitable for time-sensitive tasks.
- **Training Efficiency**: QL showed a significant advantage in training time, making it a suitable choice for scenarios where computational resources are limited. However, its performance in larger networks was less competitive.
- **Practical Implications**: The results highlight the importance of balancing computational cost and performance requirements when selecting an agent for real-world applications. For instance, QLReplay is ideal for scenarios where high performance is critical, while QL is more suitable for time-sensitive tasks.

Future research directions should focus on addressing the identified challenges through innovations in algorithm design, integration of domain knowledge, and development of more realistic simulation environments. For example, the use of Graph Neural Networks (GNNs) and Generative Adversarial Networks (GANs) could help address scalability and adaptability issues. Furthermore, the integration of large language models (LLMs) could enhance the ability of agents to learn and adapt in dynamic environments. Establishing comprehensive benchmarking platforms and improving the explainability and security of reinforcement learning models are also crucial for advancing this field. In summary, this research highlights the potential of reinforcement learning in automated penetration testing and provides a foundation for future work. By addressing the identified challenges and exploring new research directions, we can continue to drive advancements in cybersecurity technologies and contribute to the development of safer and more intelligent network systems.

NOTE: Generative AI technologies were used in this thesis for LaTeX formatting and translation.

References

[1] J. Schwartz and H. Kurniawati, "Autonomous penetration testing using reinforcement learning," *arXiv preprint arXiv:1905.05965*, 2019.

[2] Z. Hu, R. Beuran, and Y. Tan, "Automated penetration testing using deep reinforcement learning," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2020, pp. 2–10.

[3] A. Chowdhary, D. Huang, J. S. Mahendran, D. Romo, Y. Deng, and A. Sabur, "Autonomous security analysis and penetration testing," in *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*. IEEE, 2020, pp. 508–515.

[4] S. Zhou, J. Liu, D. Hou, X. Zhong, and Y. Zhang, "Autonomous penetration testing based on improved deep q-network," *Applied Sciences*, vol. 11, no. 19, p. 8823, 2021.

[5] R. Gangupantulu, T. Cody, A. Rahma, C. Redino, R. Clark, and P. Park, "Crown jewels analysis using reinforcement learning with attack graphs," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2021, pp. 1–6.

[6] K. Tran, A. Akella, M. Standen, J. Kim, D. Bowman, T. Richer, and C.-T. Lin, "Deep hierarchical reinforcement agents for automated penetration testing," *arXiv preprint arXiv:2109.06449*, 2021.

[7] Y. Li, J. Yan, and M. Naili, "Deep reinforcement learning for penetration testing of cyber-physical attacks in the smart grid," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 01–09.

[8] Y. Zhang, J. Liu, S. Zhou, D. Hou, X. Zhong, and C. Lu, "Improved deep recurrent q-network of pomdps for automated penetration testing," *Applied Sciences*, vol. 12, no. 20, p. 10339, 2022.

[9] Y. Wang, Y. Li, X. Xiong, J. Zhang, Q. Yao, and C. Shen, "Dqfd-aipt: An intelligent penetration testing framework incorporating expert demonstration data," *Security and Communication Networks*, vol. 2023, no. 1, p. 5834434, 2023.

[10] M. Li, T. Chen, H. Yan, T. Zhu, and M. Lv, "Her-pt: An intelligent penetration testing framework with hindsight experience replay," *Available at SSRN 4932007*, 2024.

[11] Y. Li, H. Dai, and J. Yan, "Knowledge-informed auto-penetration testing based on reinforcement learning with reward machine," *arXiv preprint arXiv:2405.15908*, 2024.

[12] H. V. Nguyen and T. Uehara, "Multilayer action representation based on mitre att&ck for automated penetration testing," *Journal of Information Processing*, vol. 31, pp. 562–577, 2023.

[13] Y. Yang and X. Liu, "Behaviour-diverse automatic penetration testing: A curiosity-driven multi-objective deep reinforcement learning approach," *arXiv preprint arXiv:2202.10630*, 2022.

[14] Z. Li, Q. Zhang, and G. Yang, "Eppta: Efficient partially observable reinforcement learning agent for penetration testing applications," *Engineering Reports*, p. e12818, 2023.

[15] X. Guo, J. Ren, J. Zheng, J. Liao, C. Sun, H. Zhu, T. Song, S. Wang, and W. Wang, "Automated penetration testing with fine-grained control through deep reinforcement learning," *Journal of Communications and Information Networks*, vol. 8, no. 3, pp. 212–220, 2023.

[16] E. Cengiz and M. Gök, "Reinforcement learning applications in cyber security: A review," *Sakarya University Journal of Science*, vol. 27, no. 2, pp. 481–503, 2023.

[17] G. Palmer, C. Parry, D. J. Harrold, and C. Willis, "Deep reinforcement learning for autonomous cyber operations: A survey," *arXiv preprint arXiv:2310.07745*, 2023.

[18] S. Vyas, J. Hannay, A. Bolton, and P. P. Burnap, "Automated cyber defence: A review," *arXiv preprint arXiv:2303.04926*, 2023.

[19] Q. Li, R. Wang, D. Li, F. Shi, M. Zhang, and A. Chattopadhyay, "Dynpen: Automated penetration testing in dynamic network scenarios using deep reinforcement learning," *IEEE Transactions on Information Forensics and Security*, 2024.

[20] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[21] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable markov processes over a finite horizon," *Operations research*, vol. 21, no. 5, pp. 1071–1088, 1973.

[22] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[23] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, "Learning policies for partially observable environments: Scaling up," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 362–370.

[24] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman, "Acting optimally in partially observable stochastic domains," in *Aaai*, vol. 94, 1994, pp. 1023–1028.

[25] M. D. R. Team, "Cyberbattlesim," https://github.com/microsoft/cyberbattlesim, 2021.

[26] X. Ou, S. Govindavajhala, A. W. Appel *et al.*, "Mulval: A logic-based network security analyzer." in *USENIX security symposium*, vol. 8. Baltimore, MD, 2005, pp. 113–128.

[27] V. Mnih, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[28] T.-y. Zhou, Y.-c. Zang, J.-h. Zhu, and Q.-x. Wang, "Nig-ap: A new method for automated penetration testing," *Frontiers of Information Technology & Electronic Engineering*, vol. 20, no. 9, pp. 1277–1288, 2019.

[29] H. Kurniawati, D. Hsu, and W. S. Lee, "Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces," 2009.

[30] D. C. Baulcombe, "Fast forward genetics based on virus-induced gene silencing," *Current opinion in plant biology*, vol. 2, no. 2, pp. 109–113, 1999.

[31] K. Tran, M. Standen, J. Kim, D. Bowman, T. Richer, A. Akella, and C.-T. Lin, "Cascaded reinforcement learning agents for large action spaces in autonomous penetration testing," *Applied Sciences*, vol. 12, no. 21, p. 11265, 2022.

[32] A. Confido, E. V. Ntagiou, and M. Wallum, "Reinforcing penetration testing using ai," in *2022 IEEE Aerospace Conference (AERO)*. IEEE, 2022, pp. 1–15.

[33] R. Gangupantulu, T. Cody, P. Park, A. Rahman, L. Eisenbeiser, D. Radke, R. Clark, and C. Redino, "Using cyber terrain in reinforcement learning for penetration testing," in *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE, 2022, pp. 1–8.

[34] Q. Li, M. Zhang, Y. Shen, R. Wang, M. Hu, Y. Li, and H. Hao, "A hierarchical deep reinforcement learning model with expert prior knowledge for intelligent penetration testing," *Computers & Security*, vol. 132, p. 103358, 2023.

[35] J. Yi and X. Liu, "Deep reinforcement learning for intelligent penetration testing path design," *Applied Sciences*, vol. 13, no. 16, p. 9467, 2023.

[36] Q. Li, M. Hu, H. Hao, M. Zhang, and Y. Li, "Innes: An intelligent network penetration testing model based on deep reinforcement learning," *Applied Intelligence*, vol. 53, no. 22, pp. 27 110–27 127, 2023.

[37] A. Samad, S. Altaf, and M. J. Arshad, "Advancements in automated penetration testing for iot security by leveraging reinforcement learning," *evaluation*, vol. 8, p. 9, 2024.

[38] R. S. Sutton, "Reinforcement learning: An introduction," *A Bradford Book*, 2018.

[39] H. V. Nguyen and T. Uehara, "Hierarchical action embedding for effective autonomous penetration testing," in *2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*. IEEE, 2022, pp. 152–157.

[40] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," *arXiv preprint arXiv:1512.07679*, 2015.

[41] H. V. Nguyen, H. N. Nguyen, and T. Uehara, "Multiple level action embedding for penetration testing," in *Proceedings of the 4th International Conference on Future Networks and Distributed Systems*, 2020, pp. 1–9.

[42] H. V. Nguyen, S. Teerakanok, A. Inomata, and T. Uehara, "The proposal of double agent architecture using actor-critic algorithm for penetration testing." in *ICISSP*, 2021, pp. 440–449.

[43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[44] A. Chowdhary, K. Jha, and M. Zhao, "Generative adversarial network (gan)-based autonomous penetration testing for web applications," *Sensors*, vol. 23, no. 18, p. 8014, 2023.

[45] J. Schwartz, H. Kurniawati, and E. El-Mahassni, "Pomdp+ information-decay: Incorporating defender's behaviour in autonomous penetration testing," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 235–243.

[46] A. Sychugov and M. Grekov, "Automated penetration testing based on adversarial inverse reinforcement learning," in *2024 International Russian Smart Industry Conference (SmartIndustryCon)*. IEEE, 2024, pp. 373–377.

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

[48] M. C. Ghanem, T. M. Chen, and E. G. Nepomuceno, "Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks," *Journal of Intelligent Information Systems*, vol. 60, no. 2, pp. 281–303, 2023.

[49] K. Qian, D. Zhang, P. Zhang, Z. Zhou, X. Chen, and S. Duan, "Ontology and reinforcement learning based intelligent agent automatic penetration test," in *2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*. IEEE, 2021, pp. 556–561.