

Autonomous Penetration Testing Framework Based on Reinforcement Learning and Intrinsic Motivation

Rasha A. Alsakran, Sultan S. Alqahtani

Computer & Information Sciences College,
Al Imam Mohammad Ibn Saud Islamic University, Riyadh, Saudi Arabia

Abstract—Ethical penetration testing (PT) is a structured security assessment in which cybersecurity experts simulate real attacks on an organization's digital assets in a controlled testing environment. The primary objective is to evaluate the effectiveness of cybersecurity controls by detecting vulnerabilities in systems, networks, or applications. Subsequently, report these vulnerabilities to facilitate remediation prior to exploitation by malicious adversaries. In most cases, manual penetration testing relies mostly on expert knowledge and requires substantial human resources, particularly in cases where the network infrastructure is large and complex. Hence, the development of autonomous penetration testing emerges as a promising avenue. Recently, reinforcement learning (RL) has gained attention as an effective approach for automating penetration testing by enabling intelligent agents to interact with target environments and make adaptive attack decisions. This research study proposes an automated penetration testing framework termed PTiRL, powered by reinforcement learning, particularly the Deep Q-Network (DQN) algorithm. Unlike prior work, the proposed framework enhances agent learning by incorporating intrinsic motivation (IM) mechanisms, such as Random Network Distillation (RND), to address challenges associated with sparse and delayed rewards. A comparative evaluation is conducted between our PTiRL agents that utilize intrinsic motivation and those agents relying solely on standard reward signals, demonstrating the effectiveness of intrinsic motivation in improving environment exploration and learning efficiency in attack path discovery. The experiments were conducted using the Network Attack Simulator (NASim), which is an abstract network attack simulator that has benchmark scenarios designed to be used for rapid testing of autonomous penetration testing agents based on reinforcement learning.

Keywords—Framework; autonomous; penetration testing; reinforcement learning; agent; intrinsic motivation

I. INTRODUCTION

Penetration testing (PT) is a security assessment approach in which evaluators mimic real attacks to identify vulnerabilities that may bypass the defenses of a digital asset. This process often involves performing attack techniques using similar tools and methods as malicious actors, while operating within defined constraints [1]. Broadly, PT can be classified into three categories: manual, automated, and intelligent approaches [2].

Manual PT relies heavily on security professionals who utilize exploitation tools and custom scripts to trigger vulnerabilities, followed by executing payloads after successful exploitation. This approach combines specialized tools with expert judgment to carry out the testing process [3]. However,

Manual PT effectiveness depends mainly on the expertise of practitioners, and it lacks adaptability when the target environment changes. Furthermore, as systems grow in complexity, manual assessment becomes time-intensive and labor-intensive [4]. To mitigate these challenges, automated PT techniques have been introduced. The automated methods have two main categories. The first integrates outputs from vulnerability scanners with penetration testing tools, forming frameworks that automate stages such as scanning, exploitation, payload selection, and post-exploitation. Despite this integration, critical decisions such as choosing appropriate exploits often still require human intervention. Consequently, these systems remain dependent on predefined rules designed by experts and lack the ability to adapt to dynamic environments [4]. The second category models the penetration process using attack graphs, which represent possible attack paths and their consequences. By leveraging known relationships between vulnerabilities and exploits, these graphs assist in identifying feasible attack sequences [5]. However, attack graphs can generate common attack sequences, but as the attack progresses, previously identified paths may no longer be valid based on newly discovered system states. Therefore, attack graph reliance on prior knowledge of the target system limits their applicability since environments are partially unknown. To overcome these limitations, Artificial Intelligence (AI) has emerged in the cybersecurity field [6]. Among these AI branches, Reinforcement Learning (RL) based methods are particularly prominent. RL enables an agent to learn optimal behavior through interaction with an environment [7]. The agent observes the environment, takes actions, and receives feedback in the form of rewards, gradually learning a policy that maximizes its cumulative rewards over time. In the context of penetration testing, observations may correspond to system states or identified vulnerabilities, while actions represent potential exploits or attack steps. Successful attacks yield a reward, guiding the agent toward effective strategies [8].

One of the main challenges in reinforcement learning is that traditional agents rely mainly on extrinsic rewards provided by the environment, where agents receive feedback only after achieving predefined attack objectives. Such reward structures may be sparse or delayed, which limits exploration efficiency and slows the learning process, especially in large attack environments [9].

Intrinsic Motivation (IM) has emerged as a promising solution to overcome these challenges by encouraging the agent to explore and learn beyond external rewards. IM is particularly valuable for handling sparse reward environments, improving

meaningful state representation learning, supporting temporal abstraction through high-level actions, and enabling automatic learning without extensive expert intervention [9].

Furthermore, a review of the existing literature reveals limited investigation into the use of intrinsic motivation RL agents for autonomous penetration testing, despite their demonstrated effectiveness in other intrinsically motivated RL-agent domains such as in A GridWorld game by a Google researcher [10], Atari video games by an OpenAI researcher [11], and Traffic Signal Control domains [12].

In this study, we will propose an autonomous penetration testing framework based on reinforcement learning & intrinsic motivation termed PTiRL. Overall, our study made the following contributions:

- Determine and analyze the limitations associated with applying reinforcement learning to the autonomous penetration testing domain.
- Propose a framework that integrates reinforcement learning with intrinsic motivation to enhance autonomous penetration testing through improved exploration and attack path discovery.
- Conduct a case study to evaluate the intrinsic motivation impacts on the agent's performance in both training and evaluation phases compared to a standard extrinsic reward mechanism.

The study is structured as follows: in Section II, we introduce the background information of our research. Next, we present related work in Section III. We then present the proposed framework for autonomous penetration testing based on reinforcement learning and intrinsic motivation in Section IV. Following this, we describe the experimental phases and discuss the results in Section V. Finally, Section VI contains our study's future directions and conclusions.

II. BACKGROUND

This section will discuss the core concepts related to autonomous penetration testing based on reinforcement learning and intrinsic motivation, critical discoveries in this field, and contemporary publications that address the topic.

A. Penetration Testing

Based on the Penetration Testing Execution Standard (PTES) [13], penetration testing is typically structured into seven phases that simulate real-world adversary behavior in a controlled and ethical manner. The first phase is information gathering, which involves passive reconnaissance to collect public and technical intelligence about the target environment. This is followed by the scanning and enumeration phase, where live hosts, open ports, services, and configuration details are identified. The third phase is threat modeling, during which the collected data are correlated with known vulnerabilities to determine potential attack vectors. Next is the exploitation phase, which aims to validate and demonstrate the severity of identified vulnerabilities through controlled attacks. If exploitation is successful, the process moves to the post-exploitation phase, where the assessor evaluates the extent of access gained, including privilege escalation, data exposure, and

persistence opportunities. The sixth phase is reporting, which documents the findings, associated risk ratings, proof of concept evidence, and recommended remediation actions. Finally, the remediation and retesting phase ensures that the identified vulnerabilities have been effectively addressed [13].

In addition, penetration testing can be conducted across multiple domains, including web applications, APIs, mobile applications, networks and infrastructure, host and server configurations, cloud environments, IoT and embedded systems, wireless networks, and physical security, as well as within red team engagements, depending on organizational requirements [3].

Moreover, regarding penetration testing target environments, Janisch et al. [14] proposed a categorization based on the nature of the environment, which are: simulated environment, emulated environment, and real network environment. While a simulated environment refers to an abstracted and controlled setting in which the network and its components are mathematically or logically modeled without relying on real systems, an emulated environment represents a more realistic setup where virtualized systems and services replicate the behavior of actual networks with high accuracy. In addition, a real network environment denotes testing conducted on a real system, where actual infrastructure, configurations, and user interactions are involved, thereby providing the highest level of realism but also introducing greater risk and ethical considerations [14].

Furthermore, there are multiple attack techniques conducted by penetration testers, including: initial access, execution, persistence, privilege escalation, defense evasion, credential access, discovery, lateral movement, command and control, and exfiltration [15]. Initial access refers to the methods used by an attacker to gain an initial foothold within a target system or network, such as phishing or exploiting public applications. Execution is the process through which malicious code is run on a compromised system to achieve attacker objectives. In addition, persistence is the set of techniques that enable an attacker to maintain long-term access to a system despite reboots or defensive actions. While privilege escalation involves gaining higher-level permissions to access restricted resources or functionalities, defense evasion includes techniques used to avoid detection by security controls such as antivirus or intrusion detection systems. In addition, credential access refers to techniques aimed at obtaining account credentials, including passwords and authentication tokens. While discovery involves gathering information about the system, network, and environment to understand potential attack paths, lateral movement refers to techniques used to move across systems within a network after initial compromise. For command and control, it describes the mechanisms through which attackers communicate with compromised systems to issue commands and receive data. Exfiltration refers to the unauthorized transfer of sensitive data from the target environment to an external location [15]. Moreover, penetration testing methodologies vary based on the level of knowledge provided to the tester. Black box testing simulates an external attacker with no prior knowledge of the system, white box testing provides full disclosure such as system architecture and configurations, and gray box testing offers partial knowledge to enable more focused

and realistic assessments. The choice of methodology depends on the organization's objectives, threat model, and available resources [3].

B. Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning, which is itself a subset of artificial intelligence [7]. As early as 1959, Arthur Samuel, one of the pioneers of machine learning, defined the field as enabling computers to learn without explicit programming. Based on different feedback mechanisms, Zhou Zhihua categorized machine learning into three main types: supervised learning, unsupervised learning, and reinforcement learning [16].

Reinforcement learning is a learning paradigm in which an agent improves its behavior by reinforcing effective actions and discouraging ineffective ones through evaluative feedback. An RL system typically consists of the following components:

- Agent: the entity being trained, typically represented by an algorithm and its parameters.
- Environment: the external system with which the agent interacts.
- State: the current situation or observation perceived by the agent.
- Action: the decision or behavior taken by the agent in each state.
- Episode: a complete sequence of interactions from an initial state to a terminal state (success or failure).
- Step: a single interaction within an episode.
- Value: the expected benefit of taking a particular action.

The interaction between the agent and the environment forms the foundation of reinforcement learning. The agent observes the current state, selects an action, and receives a new state along with a corresponding reward from the environment. The primary objective is to learn a policy that maximizes long-term cumulative rewards by estimating the value of different state-action pairs [16].

The reinforcement learning algorithms are classified based on the following dimensions: Markov decision process (MDP) versus partially observable Markov decision process (POMDP), model-free reinforcement learning (MFRL) versus model-based reinforcement learning (MBRL), off-policy versus on-policy learning methods [17]. Where MDP represents a fully observable framework in which the agent has complete knowledge of the environment's state, while POMDP models scenarios with partial observability, where the agent must make decisions based on incomplete or uncertain state information. In addition, model-free RL refers to approaches that learn optimal policies directly from interactions without explicitly modeling the environment dynamics, whereas model-based RL constructs or utilizes an internal model of the environment to support planning and decision-making. While off-policy learning methods learn a target policy independently from the behavior policy used to generate data, on-policy methods learn exclusively from actions taken under the current policy [17].

RL model environment settings rely on the following dimensions: action space, state space, time horizon, and environment characteristics. The action space refers to the set of possible actions available to the agent, which may be discrete or continuous, as well as finite or infinite depending on the problem formulation. In addition, the state represents all possible states of the environment, which can similarly be discrete or continuous, and either finite or infinite in scale. While the time horizon defines the temporal scope of decision making, it may be finite, where interactions occur over a limited number of steps, or infinite, where the process continues indefinitely. In addition, the environment 'transition model' describes how states evolve in response to actions, which may be stationary when the underlying dynamics remain constant over time, or non-stationary when the environment changes, requiring the agent to continuously adopt its policy [17] [18].

In addition, reinforcement learning has the following characteristics: trial-and-error search, delayed reward, and environmental terrain. Trial-and-error search refers to the process by which an agent interacts with its environment by exploring different actions and learning from the outcomes to identify optimal behaviors. In addition, delayed rewards denote the scenario in which feedback is not immediately received after an action, requiring the agent to optimize long-term cumulative rewards rather than short gains. Terrain represents the structural properties of the environment, including obstacles and constraints that influence the agent's decision and navigation strategy [18].

C. Intrinsic Motivation (IM)

IM aims to encourage an agent to exhibit certain behaviors without relying solely on direct feedback from the environment. In simple terms, it refers to performing actions for inherent satisfaction rather than for externally assigned rewards. This concept is inspired by developmental learning, where humans naturally acquire skills by exploring and interacting with their surroundings [21]. The activity is considered intrinsically motivating for an autonomous agent when its interest is driven mainly by comparing or processing information obtained from different stimuli, independent of their semantic meaning. In this context, the agent does not require prior knowledge about the meaning of the observations it receives. Instead, motivation emerges from the agent's ability to detect novelty, uncertainty, or changes in information. In contrast, extrinsic rewards are generated directly by the environment through predefined functions and are not influenced by the agent's previous experiences [9].

Computational approaches to intrinsic motivation are generally classified into three major categories:

1) *Knowledge-based approaches*: These methods emphasize environment exploration and include prediction-based, novelty-based, information-based, and learning progress techniques. They are influenced by theories such as cognitive dissonance, drive theory, and optimal incongruity theory.

2) *Competence-based approaches*: These approaches motivate the agent to improve its performance by learning actions that help achieve self-generated goals. Their foundations are related to the theory of effect and flow theory.

3) *Morphological approaches*: These methods are based on synchronicity detection concepts and encourage the agent to remain in relatively stable states according to sensory inputs and internal motivational signals [21].

III. RELATED WORK

This section discusses the existing approaches and methodologies proposed in recent research on autonomous penetration testing based on reinforcement learning. Several researchers have proposed numerous approaches to improve the effectiveness and automation of the penetration testing process through the implementation of different mechanisms.

Zhenguo et al. [22] propose an automated penetration testing framework using Deep Q-Learning, where network topology is generated using Shodan and MulVAL attack graphs, with the goal of finding all possible attack paths and building a matrix representation as needed then it discovers the easiest-to-exploit attack path from the possible candidates. The study used a sparse reward mechanism, where the reward corresponding to exploiting each vulnerability is computed based on the Common Vulnerability Scoring System (CVSS).

Hoang, L. et al. [19] investigated the performance of Deep Reinforcement Learning (A3C model) in developing an automated penetration testing tool. By integrating with the Metasploit Framework, the model performs scanning and exploitation while accumulating experience over time. The authors also developed dedicated vulnerability environments and conducted various test scenarios to comprehensively evaluate the tool's performance, focusing on training and testing metrics based on achieved exploitation efficiency. It defined an extrinsic reward as positive when the selected payload was successfully implemented and negative when the payload failed.

Nguyen et al. [25] present PenGym, a framework for training penetration testing based on RL agents in realistic network environments. The framework builds an environment itself as a training platform for RL agents. Rather than developing a specific model for implementation, its primary purpose is to provide a flexible training environment for RL agents to perform penetration testing. It defines rewards in the scenario file, where positive values are added or subtracted based on the success or failure of each action.

Clintswood et al. [23] conducted a comparison between manual penetration testing and machine learning approaches. The study specifically utilized the Shennina tool with the Nmap network scanner. The primary objective was to assess the performance of human manual penetration testing versus reinforcement learning-based capabilities.

Janisch, J. et al. [14] introduced a deep reinforcement learning framework that leverages both a simulator and an emulator with a shared interface. This approach allows agents to be trained in simulation and deployed in the emulator, thereby verifying the realism of the used abstraction. It relays on sparse rewards mechanism that apply a small negative reward for each step, while positive rewards were given only when privileged access to a sensitive host was obtained.

Chen et al. [20] propose the GAIL-PT framework based on Adversarial Imitation Learning, which utilizes an expert

knowledge base to guide the policy generation of RL agents to lower the training costs. It constructs the expert knowledge bases by collecting state-action pairs from the successful exploitation of pretrained RL models. Then it feeds the expert knowledge bases generated by different RL models online into the discriminator of GAIL-PT to guide its training process. It uses an extrinsic reward mechanism based on positive and negative rewards, where successful post-exploitation resulted in the maximum reward score, successful exploitation without obtaining root authority resulted in the lowest reward score, and failed exploitation or other operations resulted in a negative reward score.

As discussed above, most of the reviewed autonomous penetration testing studies based on reinforcement learning rely primarily on standard extrinsic reward mechanisms defined by the environment, where rewards are assigned according to predefined attack outcomes such as successful exploitation, privilege escalation, or sensitive host compromise. Despite the diversity of RL algorithms and experimental environments, the reviewed studies did not incorporate intrinsic motivation techniques to encourage exploration under sparse or delayed reward conditions. This observation highlights a potential research gap in improving exploration efficiency within automated penetration testing environments.

IV. PTiRL FRAMEWORK

This section demonstrates how penetration testing can be formulated as a reinforcement learning problem within our proposed PTiRL framework. First, we define the reinforcement learning model used in this study, which is the Deep Q-Network (DQN). Next, we incorporate an intrinsic motivation (IM) mechanism, specifically the Random Network Distillation (RND) method, to enhance the agent's exploration capability. We then present the PTiRL agent pseudocode algorithm and explain its operation. Following this, we provide a detailed description of the target simulated environment, including the scenario construction and its key components, such as the configured penetration tasks and network topology. Fig. 1 illustrates the framework, architecture, and workflow.

A. Deep Q-Network (DQN)

DQN is a value-based reinforcement learning algorithm that combines Q-learning with deep neural networks to enable agents to learn optimal decision policies in complex environments with large state spaces [18]. In our framework, DQN is utilized as the core learning model for both the baseline agent and the proposed PTiRL agent.

The agent interacts with the NASim environment [26] by observing the current network state, selecting penetration testing actions, and receiving extrinsic rewards based on the outcomes of these selected actions. In regard to extrinsic rewards, the DQN model estimates the expected cumulative reward, known as the Q-value, for each possible action and gradually learns an effective penetration strategy through repeated interaction with the environment. To improve training stability, the implementation employs experience replay memory and a target network mechanism, which are standard components of the DQN architecture [18]. In addition, we introduced within PTiRL

agent an intrinsic rewards mechanism which will be illustrated in Section B.

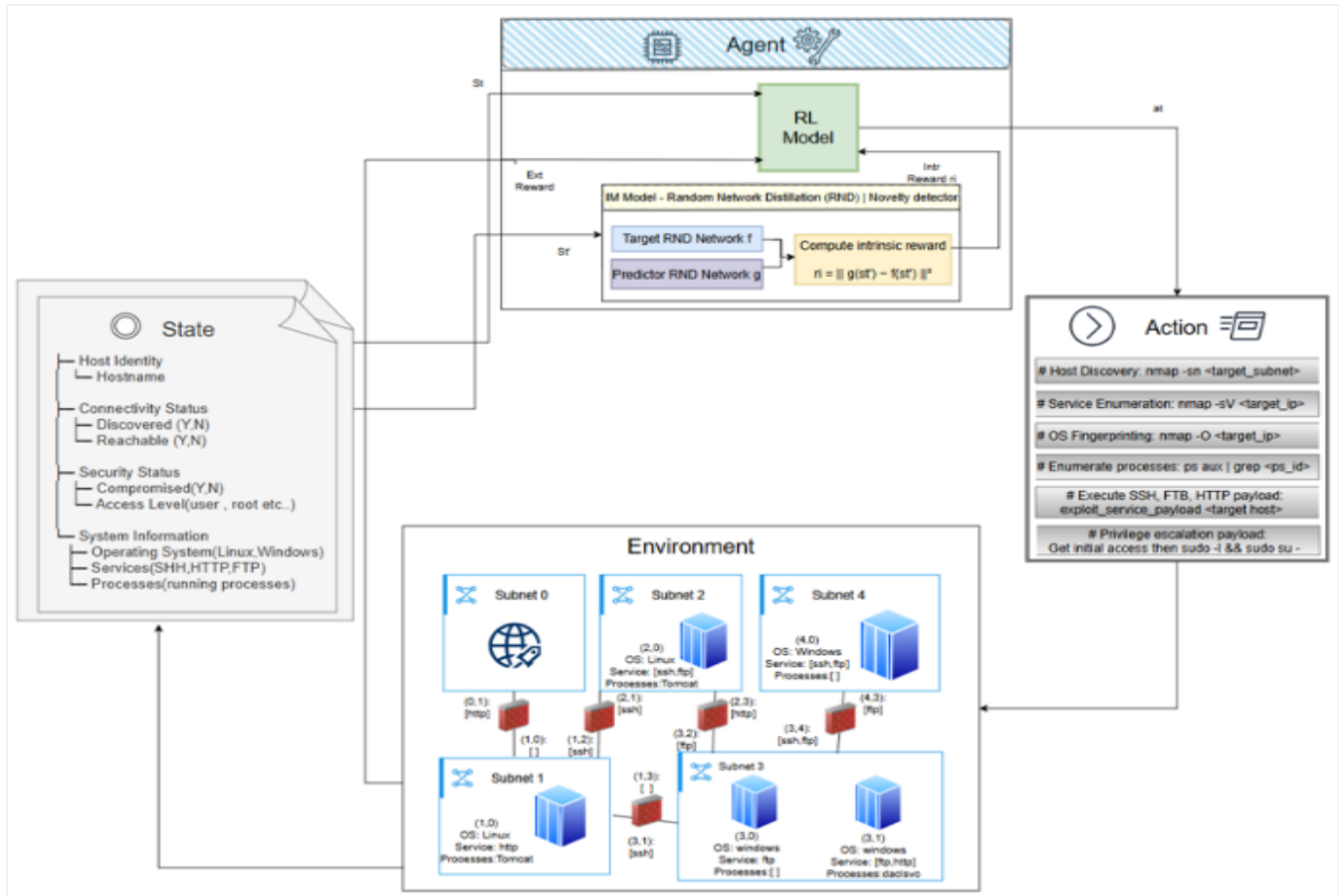


Fig. 1. PTiRL framework architecture and workflow.

B. Random Network Distillation (RND)

Intrinsic motivation in reinforcement learning refers to an approach in which an agent is driven by internally generated rewards based on curiosity, novelty, or the desire to acquire new knowledge, rather than relying on external standard rewards from the designed environment [21]. This approach aims to encourage effective exploration, particularly in environments characterized by sparse or delayed reward signals [9].

Random Network Distillation (RND) is an intrinsic motivation method that augments reinforcement learning agents with an additional reward signal for visiting novel states. It operates using two neural networks: a fixed randomly initialized target network and a trainable predictor network. The intrinsic reward is computed as the prediction error between the outputs of these two networks. The predictor network learns to mimic the target network on states it has seen frequently; the prediction error is high for unfamiliar states, which release a reward signal based on the novelty. This encourages the agent to explore previously unseen regions of the environment [21].

C. Algorithm1: PTiRL Agent

Algorithm 1 presents the overall operations of the proposed PTiRL agent based on the Deep Q-Network (DQN) algorithm integrated with an intrinsic motivation mechanism (Random

Network Distillation (RND) The agent interacts with the NASim environment by observing network states, selecting penetration testing actions, and receiving both intrinsic and extrinsic reward signals to gradually learn an effective attack strategy through continuous training and exploration.

D. Target Simulated Network Environment (NASim)

NASim is an open-source high-level network attack simulator implemented in Python and designed to support the rapid development and evaluation of autonomous penetration testing agents based on reinforcement learning [26]. It provides benchmark scenarios that define network topologies with different levels of complexity, including hosts, services, processes, vulnerabilities, and firewall configurations.

Algorithm 1: PTiRL agent

Input

Environment E (NASim)
Q-network Q0
Target network Q0-
Replay buffer D
RND target network f (fixed)
RND predictor network g (trainable)
Beta weight β 0.01
RND dimension 64
Discount factor γ

```
Exploration rate  $\epsilon$ 
Initialize
  Initialize Q-network randomly
  Initialize replay buffer D
  Initialize RND networks (f fixed, g trainable)

For each episode do:
  Reset environment and get state s

  While (episode not finished) do:
    Select action a:
      With probability  $\epsilon > \text{random action}$ 
      Otherwise  $a = \text{argmax } Q\theta(s, a)$ 
    Execute action a
    Observe next state  $s'$ , external reward  $re$ , done
    Compute intrinsic reward:
       $ri = \|g(s') - f(s')\|^2$ 
    Compute total reward:
       $r = re + \beta * ri$ 
    Store transition  $(s, a, r, s')$  in buffer D
    Sample a batch from D
    Update Q-network:
       $y = r + \gamma * \max Q\theta(s', a')$ 
      Minimize  $(y - Q\theta(s, a))^2$ 
    Update RND predictor:
      Minimize  $\|g(s') - f(s')\|^2$ 
    Update state:
       $s < s'$ 
    End While
  Periodically update target network
End For

Output:
Trained PTiRL policy
```

Each scenario is specified using structured YAML configuration files that describe the network structure, available attack actions, and target objectives. The benchmark suite includes multiple predefined scenarios (tiny, small, medium, and large) with network topologies, which enable consistent comparison by providing environments of increasing scale and difficulty. NASim scenarios have been utilized in more than 38 research studies within the same penetration testing-based RL field, such as [14], [20], [24], and [25], demonstrating their widespread adoption. The following section presents the benchmark scenarios (Tiny, Small, and Medium) of simulated networks provided by the NASim framework. The tiny network consists of 4 subnets, 4*14 observations, 576 states, and 18 action spaces, while the small network consists of 5 subnets, 9*23 observations, 24576 states, and 72 action spaces. Moreover, the medium network consists of 6 subnets, 17*27 observations, 393216 states, and 192 action spaces as presented in Table I.

TABLE I. SELECTED NASIM BENCHMARK SCENARIOS

Scenario	Subnet	Host	Action	Observation Dims	State	Step Limit
Tiny	4	3	18	4*14	576	1000
Small	5	8	72	9*23	24576	1500
Medium	6	16	192	17*27	393216	2000

The network scenarios are constructed to represent a simplified network environment by combining host configurations, attack actions, and firewall rules. Table II presents the host-level configuration; each machine is defined by its operating system, such as Linux or Windows, running services like FTP, SSH, SMTP, Samba, or HTTP, processes such as schtask, tomcat, and daclsvc, in addition to the sensitivity value 100, which represents critical targets for the attacker [26]

TABLE II. HOST LEVEL CONFIGURATION IN EACH NETWORK SCENARIO

Host Level configuration				
Tiny network				
Host	Value	Service	Process	OS
(1, 0)		ssh	tomcat	Linux
(2, 0)	100	ssh	tomcat	Linux
(3, 0)	100	ssh	tomcat	Linux
Small network				
(1, 0)		http	None	Linux
(2, 0)	100	ssh, ftp	tomcat	Linux
(3, 0)		ftp	None	Windows
(3, 1)		ftp, http	daclsvc	Windows
(3, 2)		ftp	daclsvc	Windows
(3, 3)		ftp	None	Windows
(3, 4)		ftp	daclsvc	Windows
(4, 0)	100	ssh, ftp	tomcat	Linux
Medium network				
(1, 0)		http	None	Linux
(2, 0)	100	smtp	schtask	Windows
(3, 0)		ftp	schtask	Windows
(3, 1)		ftp, http	daclsvc	Windows
(3, 2)		ftp	None	Windows
(3, 3)		ftp	schtask	Windows
(3, 4)		ftp	schtask	Windows
(4, 0)		ssh	None	Linux
(4, 1)		ssh	None	Linux
(4, 2)		ssh	None	Linux
(4, 3)		ssh, ftp	tomcat	Windows
(4, 4)		ssh, ftp	tomcat	Windows
(5, 0)	100	ssh, samba	None	Linux
(5, 1)		ssh, http	tomcat	Linux
(5, 2)		ssh	None	Linux

The action set in Table III below specifies the available penetration testing actions, such as FTP, SSH, SMTP, Samba, HTTP, Schtasks, Tomcat, and DaclSvc exploits, including available privilege escalation techniques. Each action corresponds to its applicable OS type and the required user access level for successful exploitation. In addition, each action is associated with a cost and a probability of success. These attributes define the characteristics of each available attack action. The defined actions represent different methods that can be performed across the network environment. Collectively, they provide a structured set of actions for interacting with the available hosts and services.

TABLE III. ACTION SET AVAILABLE FOR PENETRATION TESTING EXPLOITS

Action set configuration					
Tiny network					
Action	Cost	Prob	Type	Access	OS
ssh_Exploit	1	0.8	Exploit	User	Linux
tomcat_Exploit	1	1	Privilege Escalation	Root	Linux
Small network					
ssh_Exploit	3	0.9	Exploit	User	Linux
ftp_Exploit	1	0.6	Exploit	User	Windows
http_Exploit	2	0.9	Exploit	User	Any / None
tomcat_Exploit	1	1	Privilege Escalation	Root	Linux
daclsvc_Exploit	1	1	Privilege Escalation	Root	Windows
Medium network					
ssh_Exploit	3	0.9	Exploit	User	Linux
ftp_Exploit	1	0.6	Exploit	Root	Windows
http_Exploit	2	0.9	Exploit	User	Any / None
Samba_Exploit	2	0.3	Exploit	Root	Linux
Smtpt_Exploit	3	0.6	Exploit	User	Windows
Tomcat_Exploit	1	1	Privilege Escalation	Root	Linux
Daclsvc_Exploit	1	1	Privilege Escalation	Root	Windows
Schtask_Exploit	1	1	Privilege Escalation	Root	Windows

Additionally, the firewall policies in Table IV regulate communication between hosts by allowing specific ports and restricting others. These policies help define the communication structure and connectivity between different components of the network. Such configurations contribute to representing common network conditions and interactions within the simulated environment.

TABLE IV. FIREWALL TRAFFIC POLICY CONFIGURATION

Firewall Policy	
Tiny Network	
Host / Subnet	Configuration
(0, 1)	SSH
(1, 0)	none
(1, 2)	none
(2, 1)	SSH
(1, 3)	SSH
(3, 1)	SSH
(2, 3)	SSH
(3, 2)	SSH
Small network	
(0, 1)	HTTP
(1, 0)	None
(1, 2)	SSH

(2, 1)	SSH
(1, 3)	None
(3, 1)	SSH
(2, 3)	HTTP
(3, 2)	FTP
(3, 4)	SSH, FTP
(4, 3)	FTP
Medium network	
(0, 1)	HTTP
(1, 0)	None
(1, 2)	SMTP
(2, 1)	SSH
(1, 3)	None
(3, 1)	SSH
(2, 3)	HTTP
(3, 2)	SMTP
(3, 4)	SSH, FTP
(4, 3)	FTP, SSH
(3, 5)	SSH, FTP
(5, 3)	FTP, SSH

V. EXPERIMENT AND RESULT

This section presents the experiment conducted to interact with the proposed framework. The experiments were designed to compare the proposed PTiRL agent against the baseline DQN agent under identical training conditions. Initially, both agents were trained using the same benchmark scenario to evaluate the impact of integrating intrinsic motivation on the agent's learning and exploration behavior. Throughout the experiments, multiple training and evaluation metrics were recorded, including rewards, execution steps, success rate, and agent behavior progression using TensorBoard visualization tools.

A. Experiment Setting

1) *Hardware and deployment environment:* All experiments were conducted within Ubuntu 20.04.6 LTS virtual machines running on VMware Workstation. The virtual machines were allocated 2 virtual CPU cores, 4 GB of RAM, and 40 GB of disk space. The underlying physical host system was equipped with an Intel(R) Core(TM) i7-10510U processor (4 cores and 8 logical processors), 8 GB of RAM, and an NVIDIA GeForce MX330 GPU.

2) *Software requirements:* The proposed PTiRL framework was implemented in Python 3.8.10. The NASim open-source framework was used as the penetration testing simulation environment. NASim relies on several software libraries, including Gym for environment interaction, NumPy for numerical computations, and PyYAML for scenario configuration management. The reinforcement learning components and neural network models were implemented using the PyTorch framework. TensorBoard was employed to monitor the training process and visualize performance metrics.

3) *Environment Initialization:* The benchmark scenarios were initialized using partially observable mode (fully_obs = False), flat action space representation (flat_actions = True), and flat observation representation (flat_obs = True). These settings were selected to simulate a more realistic penetration

testing environment. The replay buffer stores the terminal flag (done) for each transition. During Q-network updates, the Bellman target uses this flag to prevent bootstrapping from terminal states. The implementation also distinguishes task termination from episode truncation caused by the environment step limit or the remaining training budget. Truncated episodes stop further interactions, while transitions with (done = False) retain the bootstrapped target calculation, consistent with the original NASim DQN baseline implementation [26].

In Table V, we present the hyperparameter configurations used for training and evaluating both the baseline agent and the proposed PTiRL agent. These parameters were selected to ensure stable learning behavior and fair comparison.

TABLE V. LIST OF HYPERPARAMETER VALUES OF ALGORITHMS

Network	Tiny Network		Small Network		Medium network	
Agent	PTiRL agent	Baseline agent	PTiRL agent	Baseline agent	PTiRL agent	Baseline agent
Learning rate	0.001	0.001	0.001	0.001	0.001	0.001
Discount factor	0.99	0.99	0.99	0.99	0.99	0.99
Replay buffer size	100,000	100,000	100,000	100,000	100,000	100,000
Batch size	32	32	64	64	128	128
Target update frequency	1000	1000	1000	1000	1000	1000
Training steps	20,000	20,000	150,000	150,000	250,000	250,000
Final epsilon	0.05	0.05	0.05	0.05	0.05	0.05
Hidden layers	[64,64]	[64,64]	[128,128]	[128,128]	[256,256]	[256,256]
Seeds	0	0	0	0	0	0
exploration_steps	10,000	10,000	50,000	50,000	150,000	150,000

B. Agents Training

Both the baseline DQN and the proposed PTiRL agents were trained on the NASim tiny, small, and medium network scenarios. To maintain a fair comparison within each scenario, both agents were trained using identical DQN hyperparameters. A learning rate of 0.001, discount factor of 0.99, replay buffer size of 100,000, target network update frequency of 1,000 steps, initial epsilon of 1.0, and final epsilon of 0.05 were maintained across all experiments. However, the training capacity was increased according to the complexity of the network scenario. For the Tiny network, both agents were trained for 20,000 steps using a batch size of 32, two hidden layers of [64,64], and 10,000 exploration steps. For the small network, the training budget was increased to 150,000 steps, with a batch size of 64, hidden layers of [128,128], and 50,000 exploration steps. Finally, the medium network was trained for 250,000 steps using a batch size of 128, hidden layers of [256,256], and 150,000 exploration steps. All experiments used seed 0. This scaling provided greater learning capacity and a longer exploration period as the number of hosts, available

action space, and possible attack paths increased. As illustrated in Table V, the training behavior of both agents was analyzed using four TensorBoard metrics: episode number, episode return, episode steps, and mean Q-value. In addition, the training logs uploaded in GitHub repo were examined to determine whether the penetration testing objective was reached at different stages of training. According to the TensorBoard dashboards, Fig. 2 presents the number of episodes completed during training on the tiny network. Under the same 20,000-step training budget, the PTiRL agent completed 264 episodes, compared with 179 episodes for the baseline DQN agent. Thus, PTiRL completed 85 additional episodes, approximately 47.5% more than the baseline.

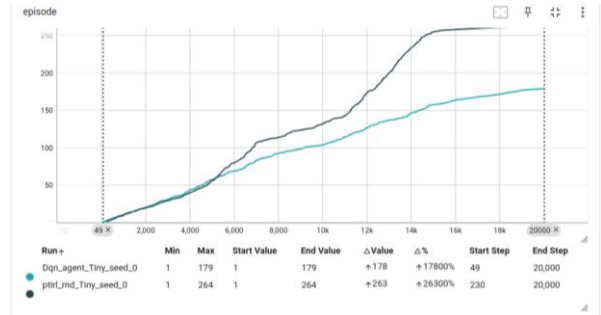


Fig. 2. Episode number during training on the tiny network.

The training logs uploaded to GitHub also show that both agents were capable of repeatedly reaching the penetration testing goal throughout a substantial portion of training. For example, the baseline DQN reported successful goal completion at the logged Episode 10 and continued to reach the goal across the subsequent logged episodes through Episode 170. Similarly, PTiRL repeatedly reached the goal from the logged Episode 10 through Episode 260. However, the final incomplete truncated episode recorded when the 20,000-step training budget was exhausted did not reach the goal for either agent. Overall, the larger number of episodes completed by PTiRL indicates that it experienced shorter episodes on average across the complete training step budget, although individual episode lengths varied considerably. In addition, Fig. 3 presents the episode returns statistics that show considerable variation in the performance of both agents throughout training. The baseline DQN recorded episode returns ranging from a minimum of -469 to a maximum of 188, whereas PTiRL recorded returns ranging from -900 to 194. These results indicate that both agents experienced episodes with relatively low returns as well as episodes with higher returns throughout the training process. PTiRL achieved a slightly higher maximum return of 194 compared with 188 for the baseline DQN, although it also recorded a lower minimum return. Overall, the variation in episode returns reflects the exploration nature of the training process, where different action selections and attack paths resulted in different rewards and costs. Moreover, Fig. 4 presents the number of steps taken per episode. The final values were very similar, at 569 steps for DQN and 562 steps for PTiRL. However, PTiRL reached a minimum episode length of only 6 steps, compared with 12 steps for DQN, while PTiRL's maximum episode length reached 1,000 steps compared with 569 for DQN. Consequently, PTiRL demonstrated both very short and very long trajectories during training. The mean Q-value statistics shown in Fig. 5 indicate

positive value function development for both agents in the tiny scenario. The baseline DQN mean Q-value increased from 1.8164 to 54.5086, while PTiRL increased from 2.8210 to 43.8344. DQN also reached a similar maximum Q-value of 81.3456, corresponding to 80.2022 for PTiRL.

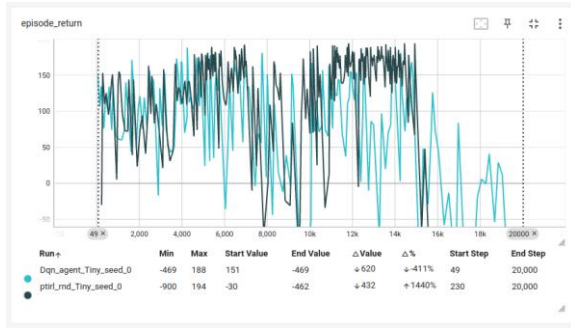


Fig. 3. Episode return during training on the tiny network.

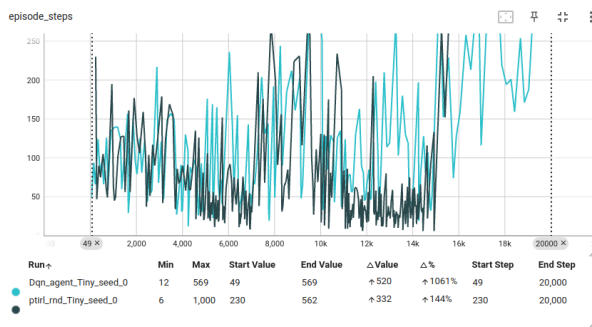


Fig. 4. Episode steps during training on the tiny network.

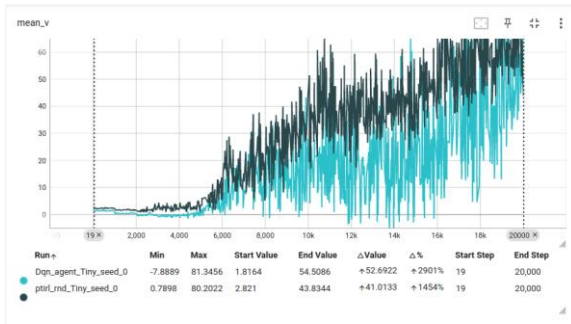


Fig. 5. Mean Q-value statistics during training on the tiny network.

Moreover, as the environment complexity increased in the small scenario, the difference between the two agents became less pronounced in terms of episode count. As illustrated in Fig. 6, PTiRL completed 155 episodes with over 150,000 interaction steps, whereas baseline DQN completed 148 episodes. This corresponds to only seven additional episodes for PTiRL, approximately 4.7% more than DQN. The training logs uploaded to GitHub also show that both agents successfully reached the goal during earlier stages of training. PTiRL reported successful goal completion at the logged Episodes 10 through 80, after which the logged Episodes 90 through 155 failed to reach the goal. Baseline DQN similarly achieved the goal during the logged Episodes 10 through 60, followed by several failures, although successful goal completion reappeared at the logged Episode 110.

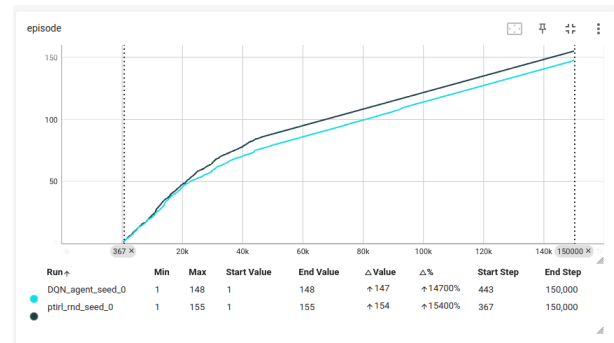


Fig. 6. Episode number during training on the small network.

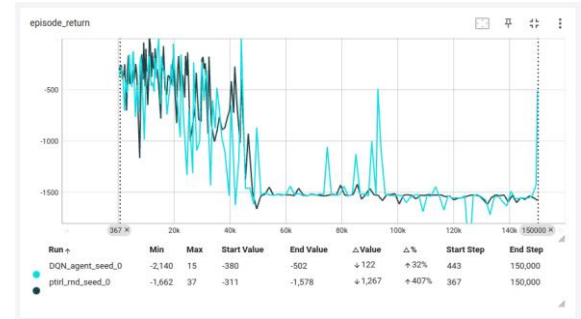


Fig. 7. Episode return during training on the small network.

In addition, Fig. 7 illustrates the variation in episode returns throughout training. Baseline DQN produced returns ranging from a minimum of -2,140 to a maximum of 15, while PTiRL produced returns ranging from -1,662 to 37. Both agents therefore experienced episodes with relatively low returns as well as episodes with substantially higher returns during training. PTiRL achieved a slightly higher maximum return of 37 than baseline DQN's 15 and also recorded a less negative minimum return of -1,662 compared with DQN's -2,140. Overall, these results demonstrate considerable variation in episode-level performance for both agents as they explored different action sequences and attack paths within the small network.

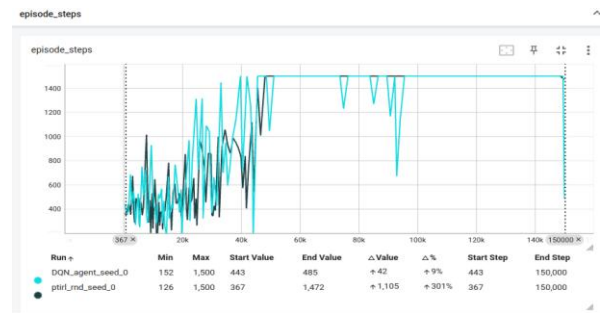


Fig. 8. Episode steps during training on the small network.

Moreover, the episode step statistics presented in Fig. 8 show variation in trajectory length for both agents throughout training. Baseline DQN recorded episode lengths ranging from a minimum of 152 steps to a maximum of 1,500 steps, while PTiRL ranged from a lower minimum of 126 steps to the same maximum of 1,500 steps. This indicates that both agents experienced a mixture of shorter and longer trajectories during

the training process. PTiRL achieved the shortest observed trajectory, requiring 126 steps compared with 152 steps for DQN. Overall, the variation in episode length reflects differences in the action sequences and exploration attack paths taken by the agents while interacting with the small network environment. In addition, the mean Q-value statistics in Fig. 9 showed that the DQN started with a mean Q-value of 2.2424 and ended at -50.7837, while PTiRL ranged from 1.8724 to -51.5590.

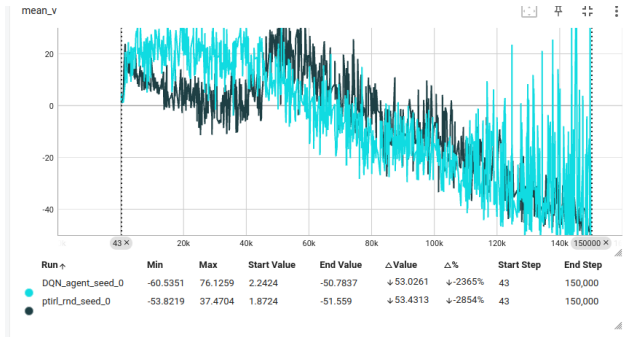


Fig. 9. Mean Q-value statistics during training on the small network.

In addition, the medium scenario represented the most complex environment evaluated in this study and was therefore allocated 250,000 training steps, 150,000 exploration steps, a batch size of 128, and two hidden layers of [256,256]. As shown in Fig. 10, the PTiRL agent completed 143 episodes compared with 139 for baseline DQN. The difference was therefore relatively small, with PTiRL completing four additional episodes, or approximately 2.9% more than baseline DQN. The training logs uploaded to GitHub also reveal that both agents successfully reached the penetration testing objective during the earlier stages of training but experienced increasing difficulty later. Baseline DQN reported successful goal completion at the logged Episodes 10 through 50, after which the logged Episodes 60 through the final Episode 139 failed to reach the goal. PTiRL reached the goal at the logged Episodes 10, 20, and 30, failed at Episodes 40 and 50, recovered successful goal completion at Episodes 60 and 80, and subsequently failed in the logged episodes through the final Episode 143.

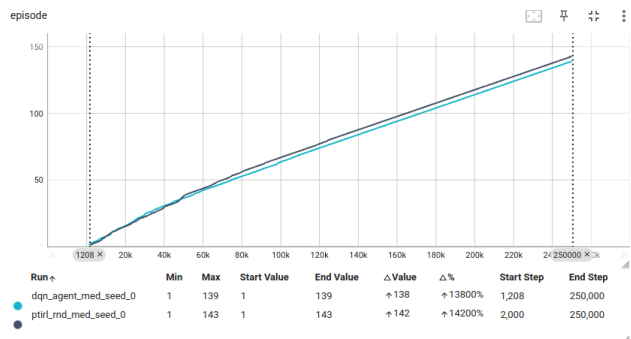


Fig. 10. Episode number during training on the medium network.

In addition, Fig. 11 shows a clear difference in episode return development. According to the TensorBoard statistics, baseline DQN started with a return of -1,586 and ended at -2,121, whereas PTiRL improved from -2,910 to -1,341.

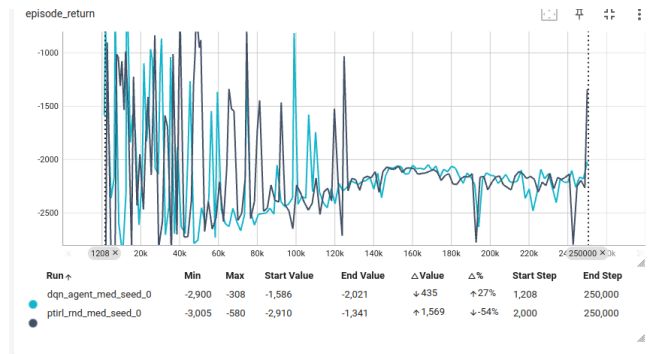


Fig. 11. Episode return during training on the medium network.

PTiRL also achieved a substantially better maximum return of -580 compared with -1,586 for baseline DQN. Although the returns remained negative, a less negative return represents improved performance because fewer accumulated action costs are incurred relative to more negative trajectories. Therefore, the medium scenario provides evidence that PTiRL developed comparatively better return behavior under the larger and more complex search space.

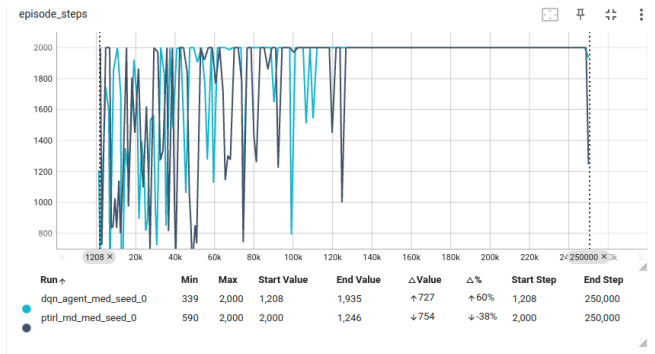


Fig. 12. Episode steps during training on the medium network.

The episode step results in Fig. 12 shows this difference. Baseline DQN increased from 1,208 steps at the beginning of the recorded series to 1,935 steps at the end, representing an increase of approximately 60%. In contrast, PTiRL decreased from 2,000 to 1,246 steps, corresponding to a reduction of approximately 38%. Both agents reached the 2,000-step episode limit during training; however, PTiRL showed a substantial reduction in episode length by the end of the training process, whereas DQN moved in the opposite direction. When considered together with the improved PTiRL episode returns, this suggests that intrinsic motivation helped the agent learn comparatively more efficient behavior in the medium scenario. Moreover, the largest difference between the two agents was observed in the mean Q-value statistics presented in Fig. 13. Baseline DQN started with a mean Q-value of 1.0992 but ended at -33.5043, with values ranging from -62.6216 to 220.9878. In contrast, PTiRL increased from 1.0004 to 1,882.0696 and reached a maximum of 2,733.0403. This substantial growth indicates that the PTiRL network developed much larger estimates of expected cumulative return than the baseline network.

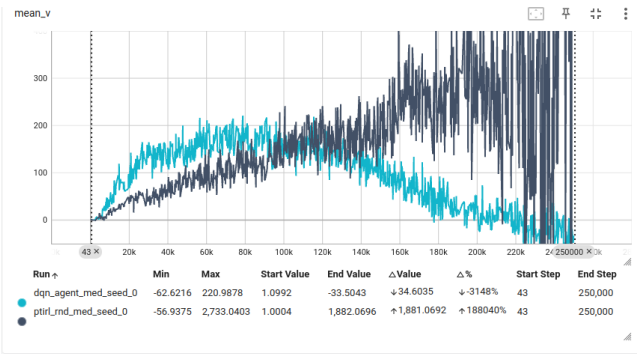


Fig. 13. Mean Q-value statistics during training on the medium network.

Overall, the training experiments demonstrate that the relative behavior of PTiRL and baseline DQN changed as network complexity increased. In the Tiny scenario, PTiRL completed substantially more episodes (264 versus 179) and achieved a slightly higher maximum return; both agents repeatedly reached the penetration testing objective during training, indicating that the relatively small environment was learnable by both approaches. In the Small scenario, however, neither agent demonstrated consistently stable late-stage behavior. PTiRL completed slightly more episodes, while the final mean Q-values of both agents were almost identical. The Medium scenario produced a different pattern. Although both agents struggled to maintain goal completion during later training, PTiRL ended with a less-negative episode return (-1,341 versus -2,121), substantially fewer episode steps (1,246 versus 1,935), and a markedly higher mean Q-value (1,882.07 versus -33.50). Thus, the benefit of intrinsic motivation became more visible in several learning metrics when the search space was enlarged, although the training logs also demonstrate that this did not guarantee consistent successful goal completion.

C. Agents Evaluation

After completing the training phase, the trained baseline DQN and PTiRL agent policies were independently evaluated on the NASim tiny, small, and medium benchmark scenarios. Each agent was evaluated over 10 episodes, which involved 31,951 recorded interactions for the baseline DQN and 29,445 interactions for the PTiRL agent. During evaluation, the selected actions, environmental observations, episode returns, number of interaction steps, and goal completion status were recorded. The evaluation used an epsilon-greedy policy, with epsilon values of 0.08 for the tiny network, 0.4 for the small network, and 0.8 for the medium network scenarios. These settings introduced increasing levels of exploratory action selection as network complexity increased. Unlike the training phase, in which epsilon decreased linearly from 1.0 to 0.05, the evaluation epsilon remained fixed within each scenario. The increasing values were used to introduce progressively greater stochastic action selection as network complexity increased, allowing the agents to be compared under different exploration conditions. Both baseline DQN and PTiRL were evaluated using identical epsilon settings within each scenario to maintain consistency in the comparative assessment. The primary evaluation for penetration testing goal achievement was the episode success rate, defined as: Episode Success Rate = Number of Successful Episodes / Total Evaluation Episodes * 100.

In addition to success rate, the number of steps required to complete an episode and the accumulated episode return were analyzed as listed in Table VI. Since actions in NASim incur costs and unsuccessful or inefficient attack actions can produce additional penalties, a less negative or higher episode return generally represents a more efficient attack trajectory. However, return was considered together with goal completion and episode length rather than being interpreted independently.

TABLE VI. EVALUATION STATISTICS ON EACH NETWORK

Evaluation statistics						
Tiny Network						
Episode/ matrix	DQN Steps	DQN Return	DQN Goal	PTiRL Steps	PTiRL Return	PTiRL Goal
Episode 1	588	-388	Yes	147	53	Yes
Episode 2	180	20	Yes	739	-539	Yes
Episode 3	1000	-900	No	537	-337	Yes
Episode 4	651	-451	Yes	1000	-900	No
Episode 5	409	-209	Yes	887	-687	Yes
Episode 6	386	-186	Yes	144	56	Yes
Episode 7	216	-16	Yes	381	-181	Yes
Episode 8	651	-451	Yes	449	-249	Yes
Episode 9	665	-465	Yes	305	-105	Yes
Episode 10	530	-330	Yes	97	103	Yes
Small Network						
Episode/ matrix	DQN Steps	DQN Return	DQN Goal	PTiRL Steps	PTiRL Return	PTiRL Goal
Episode 1	863	-765	Yes	993	-930	Yes
Episode 2	1500	-1552	No	619	-492	Yes
Episode 3	751	-649	Yes	455	-315	Yes
Episode 4	1078	-1022	Yes	758	-681	Yes
Episode 5	1397	-1379	Yes	1306	-1265	Yes
Episode 6	460	-323	Yes	1084	-1063	Yes
Episode 7	999	-930	Yes	784	-701	Yes
Episode 8	815	-723	Yes	617	-495	Yes
Episode 9	603	-494	Yes	1500	-1592	No
Episode 10	1500	-1567	No	1121	-1091	Yes
Medium network						
Episode/ matrix	DQN Steps	DQN Return	DQN Goal	PTiRL Steps	PTiRL Return	PTiRL Goal
Episode 1	795	-911	Yes	906	-1295	Yes
Episode 2	2000	-2747	No	1902	-2673	Yes
Episode 3	1423	-1780	Yes	1085	-1558	Yes
Episode 4	1853	-2388	Yes	2000	-2824	No
Episode 5	2000	-2711	No	1792	-2548	Yes
Episode 6	1965	-2580	Yes	1431	-2048	Yes
Episode 7	1330	-1667	Yes	1724	-2503	Yes
Episode 8	1412	-1769	Yes	1472	-2374	Yes
Episode 9	2000	-2715	No	2000	-2819	No
Episode 10	1931	-2487	Yes	1210	-1656	Yes

From a penetration testing perspective, reaching the goal in the tiny network scenario requires the agent to discover and compromise the target hosts and obtain the required level of access. A successful attack flow begins by exploiting the vulnerable SSH service on the initially reachable host (1,0), which provides user-level access, followed by a subnet scan that discovers the two valuable target hosts (2,0) and (3,0). The agent can then exploit the vulnerable service on these discovered hosts to obtain user-level access and subsequently use the available Tomcat privilege escalation exploit to obtain root-level access. In simple terms, the evaluated episode successfully compromised (3,0) through SSH and later escalated its access from user to root through Tomcat. Therefore, a successful episode represents more than a positive RL outcome; it demonstrates the agent's ability to perform a penetration testing sequence involving initial exploitation, network discovery, lateral movement, and privilege escalation toward the valuable hosts within the tiny network. Moreover, the small network scenario introduces a more complex attack path than the tiny network because the agent must navigate multiple subnets containing both Linux and Windows hosts and use different exploitation and privilege escalation techniques. The attack process involves gaining an initial foothold, performing network discovery to identify additional reachable hosts, exploiting exposed services such as HTTP, SSH, or FTP, and moving through intermediate systems toward the two valuable target hosts (2,0) and (4,0). Depending on the compromised host, privilege escalation can then be performed through the available Tomcat or daclsvc processes to obtain higher-level access. In the successful evaluation episodes, both valuable hosts were ultimately compromised with root-level access (Access = 2), causing the penetration testing goal to be reached. This demonstrates that success in the small scenario requires the agent to combine reconnaissance, exploitation, lateral movement across the network, and privilege escalation, rather than simply compromising a single host. In addition, the medium scenario extends this penetration testing process to a considerably larger and more diverse network. The agent must progress from the initially accessible part of the network through several intermediate subnets before reaching the valuable hosts (2,0) and (5,0), both assigned a value of 100. The environment contains Linux and Windows systems and a broader range of services, including SSH, FTP, HTTP, Samba, and SMTP, together with multiple privilege escalation opportunities. A successful attack therefore requires repeated discovery of new hosts and subnets, exploitation of appropriate services to establish footholds, movement through intermediate systems, and acquisition of the access necessary to compromise the valuable targets. Thus, the medium scenario represents a more complex multi-stage penetration test in which reconnaissance, exploitation, lateral movement, and privilege acquisition must be coordinated across a larger network topology to reach the final targets.

From a reinforcement learning perspective, on the tiny network, both baseline DQN and PTiRL achieved a 90% success rate, successfully reaching the penetration testing objective in 9 of the 10 evaluation episodes. Baseline DQN failed only in Episode 3, where it reached the 1,000-step episode limit with a return of -900. PTiRL similarly experienced one unsuccessful episode, Episode 4, which also terminated at the 1,000-step limit

with a return of -900. Despite having identical success rates, PTiRL demonstrated greater interaction efficiency. Across all 10 evaluation episodes, baseline DQN required an average of 527.6 steps per episode, whereas PTiRL required 468.6 steps, corresponding to approximately an 11.2% reduction in average interaction steps. When only successful episodes are considered, DQN required an average of approximately 475.1 steps to reach the objective, compared with 409.6 steps for PTiRL, representing a reduction of approximately 13.8%.

The fastest successful baseline DQN episode required 180 steps, while PTiRL reached the objective in only 97 steps in its fastest episode. PTiRL also produced several comparatively short successful trajectories, including Episodes 1, 6, and 10, which required 147, 144, and 97 steps, respectively. These results indicate that although both agents learned successful attack strategies, PTiRL was capable of reaching the objective using fewer environment interactions in several evaluation episodes. Episode returns show a similar pattern. Across all 10 episodes, baseline DQN achieved an average return of -337.6, compared with -278.6 for PTiRL. Considering successful episodes only, the corresponding averages were approximately -275.1 and -209.6. PTiRL also achieved the highest individual return observed in the tiny evaluation, reaching 103 in Episode 10, while DQN's highest return was 20 in Episode 2. Therefore, while both approaches achieved the same final success rate, PTiRL demonstrated better average step efficiency and return performance in the tiny scenario.

On the other hand, increasing the network complexity to the small scenario produced a clearer difference in success rate. Baseline DQN successfully reached the objective in 8 of 10 episodes, corresponding to an 80% episode success rate, whereas PTiRL succeeded in 9 of 10 episodes, achieving a 90% episode success rate. DQN failed Episodes 2 and 10, with both unsuccessful episodes reaching the 1,500 steps limit. PTiRL failed only in Episode 9, which similarly terminated at the 1,500 steps limit. PTiRL also required fewer interactions on average. Across all evaluation episodes, baseline DQN required 996.6 steps per episode, compared with 923.7 steps for PTiRL, corresponding to approximately a 7.3% reduction. When unsuccessful episodes are excluded, the difference becomes smaller: successful DQN episodes required an average of approximately 870.8 steps, while successful PTiRL episodes required approximately 859.7 steps. Thus, PTiRL's overall step advantage in this scenario resulted partly from its higher success rate, while the interaction cost of successful attack paths was relatively similar between the two agents. The fastest successful attack was also comparable between the approaches. Baseline DQN reached the objective in a minimum of 460 steps, whereas PTiRL required 455 steps in its fastest successful episode. The maximum number of steps among successful episodes was 1,397 for DQN and 1,306 for PTiRL.

In terms of return, PTiRL achieved an average of -862.5 across all evaluation episodes, compared with -940.4 for baseline DQN. However, when only successful episodes are examined, their average returns were very similar: approximately -785.6 for DQN and -781.4 for PTiRL. This indicates that the primary evaluation advantage of PTiRL in the small network was its higher probability of successfully reaching the objective rather than a substantial reduction in the

cost of successful attack trajectories. This result is particularly noteworthy when compared with the small network training statistics. During the later training stage, PTiRL exhibited a lower final episode return and considerably longer final episode length than DQN. Nevertheless, during independent evaluation, PTiRL achieved a higher episode success rate.

In addition, the medium scenario presented the greatest challenge to both agents. Baseline DQN successfully reached the penetration testing objective in 7 of 10 episodes, resulting in a 70% episode success rate. PTiRL completed 8 of 10 episodes, achieving an 80% episode success rate. Baseline DQN failed in Episodes 2, 5, and 9, with each unsuccessful episode reaching the 2,000-step limit. PTiRL failed in Episodes 4 and 9, which also reached the maximum 2,000-step episode length. Across all evaluation episodes, baseline DQN required an average of 1,670.9 steps, while PTiRL required 1,552.2 steps, corresponding to approximately a 7.1% reduction in average episode length. Considering successful episodes only, DQN required approximately 1,529.9 steps, whereas PTiRL required approximately 1,440.3 steps, representing an improvement of approximately 5.9%. These results indicate that PTiRL maintained a modest interaction-efficiency advantage even in the most complex scenario. However, unlike the tiny scenarios, PTiRL did not achieve a better average return in the medium evaluation. Across all episodes, baseline DQN obtained an average return of -2,175.5, compared with -2,229.8 for PTiRL. Among successful episodes, DQN achieved an average return of approximately -1,940.3, whereas PTiRL obtained approximately -2,081.9. Furthermore, DQN's fastest successful episode required 795 steps, compared with 906 steps for PTiRL. Therefore, the medium results present a trade-off: PTiRL achieved a higher overall episode success rate and lower average episode length, while DQN produced slightly better average returns and the fastest individual successful trajectory.

VI. FUTURE WORK AND CONCLUSION

There are some future research directions that can further extend and improve the proposed PTiRL framework. Future research may integrate other intrinsic motivation (IM) techniques, such as the Intrinsic Curiosity Module (ICM) or hybrid curiosity-based approaches, that could be integrated and compared against the Random Network Distillation (RND) mechanism to evaluate their effectiveness in autonomous penetration testing environments. Moreover, future research may investigate transferring learned penetration testing policies to emulated network environments or real-world cyber ranges to evaluate the practical applicability of reinforcement learning-based penetration testing systems beyond simulated benchmark scenarios.

To conclude our research presented PTiRL, an autonomous penetration testing framework based on reinforcement learning that integrates intrinsic motivation (IM) through the Random Network Distillation (RND) mechanism with the Deep Q-Network (DQN) algorithm. The proposed framework was evaluated within the NASim simulated penetration testing environment to investigate the impact of intrinsic motivation on agent exploration behavior and attack path learning efficiency during penetration testing. The experimental results demonstrated that the PTiRL framework achieved an

improvement in several learning and evaluation metrics compared to the baseline DQN agent under identical training and evaluation conditions. During training, the PTiRL agent completed larger number of episodes within the same training budget, achieved higher episode return values, reduced the number of execution steps required to complete penetration testing tasks, and demonstrated stronger value function estimation through higher mean Qvalues in Tiny and medium network benchmark. These observations indicate that the intrinsic motivation mechanism contributed to enhanced exploration capability and enabled the agent to discover more efficient attack paths within the NASim environment. During evaluation, the results demonstrate a general reduction in penetration testing success as network complexity increased. Baseline DQN decreased from a 90% episodes success rate in tiny to 80% in small and 70% in medium network. PTiRL maintained a 90% episodes success rate in both tiny and small before decreasing to 80% in medium network. A consistent interaction efficiency advantage was also observed for PTiRL. Its average episode length was lower than baseline DQN in all three scenarios: 468.6 versus 527.6 steps in tiny network, 923.7 versus 996.6 in small network, and 1,552.2 versus 1,670.9 in medium network. This corresponds to reductions of approximately 11.2%, 7.3%, and 7.1%, respectively. Although the size of the advantage decreased as network complexity increased. Therefore, the proposed PTiRL framework demonstrates its clearest advantage in maintaining successful penetration testing goal achievement, while the results also show that intrinsic motivation does not guarantee superior performance across every individual metric. Overall, the findings suggest that integrating intrinsic motivation into autonomous penetration testing agent based on reinforcement learning can improve attack path discovery and exploration within simulated network environments.

The implementation of the proposed PTiRL framework is maintained in a public GitHub repository at the following link: <https://github.com/rskran/PTiRL>

REFERENCES

- [1] N. I. o. S. a. T. (NIST), "Penetration Testing," NIST Glossary. [Online]., Available: https://csrc.nist.gov/glossary/term/penetration_testing.
- [2] [R. a. M. W. Simon, "A Comparison of Autonomous Penetration Testing Agents," Association for Computing Machinery, no. <https://doi.org/sdl.idm.oclc.org/10.1145/3664476.3664484>, 2024.
- [3] G.-K. U. G3-Sarker, F. Yunus and A. Deraman, Penetration Taxonomy: A Systematic Review on the Penetration Process, Framework, Standards, Tools, and Scoring Methods, <https://doi.org/10.3390/su151310471>: Emerging Technologies, Sustainable Engineering and Cybersecurity in the Digital Age, 2023.
- [4] J. Liu, Y. Zhang, S. Zhou, J. Yang, Y. Lu and X. Zhong, "Autonomous penetration testing using reinforcement learning: A review and perspectives," Expert Systems with Applications, 2026.
- [5] S. T. a. B. Wu, "Attribute attack graph generation method based on attack stage," 2nd International Conference on Electronics, Communications and Information Technology (CECIT), no. doi: 10.1109/CECIT53797.2021.00057, pp. 286-291, 2021.
- [6] M. O.-O. e. al., "A Comprehensive Survey: Evaluating the Efficiency of Artificial Intelligence and Machine Learning Techniques on Cyber Security Solutions," IEEE Access, pp. 12229-12256, 2024.
- [7] S. Alwabisi, "AI in Penetration Testing: A Systematic Mapping Study," <https://www.techrxiv.org/doi/full/10.36227/techrxiv.175099664.46246512/v1>. [Accessed 20 Feb 2026].

- [8] N. L. N. L. ., G. ., J. C. Rui Fernandes, " Autonomous Pentesting Using Reinforcement Learning: A Systematic Literature Review," 7 April 2025. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5208526. [Accessed 20 Feb 2026].
- [9] A. L. M. a. S. H. Aubret, " A survey on intrinsic motivation in reinforcement learning," arXiv preprint arXiv:1908.06976 (2019).
- [10] R. D. M. G. a. O. P. Léonard Hussenot, "Show me the Way: Intrinsic Motivation from Demonstrations," in In Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '21), Richland, SC, 620–628, 2021.
- [11] Y. E. H. S. A. a. K. O. Burda, "Exploration by random network distillation," arXiv preprint arXiv:1810.12894, 2018.
- [12] P. P. Z. L. a. Y. T. L. Zhu, " MetaVIM: Meta Variationally Intrinsic Motivated Reinforcement Learning for Decentralized Traffic Signal Control," IEEE Transactions on Knowledge and Data Engineering, vol. 35, pp. no. 11, pp. 11570-11584, 1 Nov. 2023.
- [13] G. K. e. al., "Penetration Testing Execution Standard," 2010. [Online]. Available: http://www.pentest-standard.org/index.php/Main_Page. [Accessed 20 Feb 2026].
- [14] J. Janisch, T. Pevny and V. Lisý, "NASimEmu: Network Attack Simulator & Emulator for Training Agents Generalizing to Novel Scenarios," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2024.
- [15] "MITRE ATT&CK," the Mitre Corporation, 2013. [Online]. Available: <https://attack.mitre.org/>.
- [16] P. Shengguang, "Overview of Meta-Reinforcement Learning Research," 2nd International Conference on Information Technology and Computer Application (ITCA), no. <https://doi.org/10.1109/itca52113.2020.00019>, Dec. 2020.
- [17] C. Szepesvári, Algorithms for Reinforcement Learning, 10.2200/S00268ED1V01Y201005AIM009: Morgan & Claypool Publishers, 2010.
- [18] G.-H. Z. & T. Yu, Taxonomy of Reinforcement Learning Algorithms, springer, 2020.
- [19] L. V. Hoang, N. X. Nhu, N. To Trong, N. H. Quyen, V. H. Pham and P. N. Duy, "Leveraging Deep Reinforcement Learning for Automating Penetration Testing in Reconnaissance and Exploitation Phase," in Proceedings - 2022 RIVF International Conference on Computing and Communication Technologies, RIVF 2022, 2022.
- [20] J. Chen, S. Hu, H. Zheng, C. Xing and G. Zhang, "GAIL-PT: An intelligent penetration testing framework with generative adversarial imitation learning," Computers and Security, 2023.
- [21] M. Pecháč, "Intrinsic Motivation in Reinforcement Learning," Ph.D. dissertation, Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Bratislava, Slovakia., 2023. [Online]. Available: <https://cogsci.fmph.uniba.sk/~farkas/theses/matej.pecchac.phd23.pdf>.
- [22] Z. Hu, R. Beuran and Y. Yasuo Tan, "Automated Penetration Testing Using Deep Reinforcement Learning," in Proceedings - 5th IEEE European Symposium on Security and Privacy Workshops, Euro S and PW 2020, 2020.
- [23] Clintswood, D. G. Lie, L. Kuswandana, Nadia, S. Achmad and D. Suhartono, "The Usage of Machine Learning on Penetration Testing Automation," in Proceedings - 2023 3rd International Conference on Electronic and Electrical Engineering and Intelligent System: Responsible Technology for Sustainable Humanity, ICE3IS 2023, 2023.
- [24] S. Bera, L. Glenn, A. Raghavan, E. Meno, T. Cody and P. A. Beling, "Deterring Adversarial Learning in Penetration Testing by Exploiting Domain Adaptation Theory," in 2023 Systems and Information Engineering Design Symposium, SIEDS 2023, 2023.
- [25] H. P. T. Nguyen, K. Hasegawa, K. Fukushima and R. Beuran, "PenGym: Realistic training environment for reinforcement learning pentesting agents," Computers and Security, 2025.
- [26] J. a. K. H. Schwartz, "NASim: Network Attack Simulator," schwartz2019nasim, 2019. [Online]. Available: <https://networkattacksimulator.readthedocs.io/en/latest/index.html>. [Accessed 9 Feb 2026]