Received: 1 September 2025 | Revised: 23 January 2026 | Accepted: 23 March 2026

DOI: 10.4218/etrij.2025-0365

ETRI Journal WILEY

ORIGINAL ARTICLE

E-NASim: A reinforcement-learning-based cyberattack simulation framework with pre-/post-state modeling based on MITRE ATT&CK

Ki Jong Koo[1,2] | **Daesung Moon**[1] | **Byung Chul Kim**[2] | **Jae Yong Lee**[2]

[1]Intelligent Network Security Research Section, Cyber Security Research Division, Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea

[2]Department of Radio & Information Communications Engineering, Chungnam National University, Daejeon, Republic of Korea

Correspondence
Daesung Moon, Intelligent Network Security Research Section, Cyber Security Research Division, Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea.
Email: daesung@etri.re.kr

Funding information
Institute of Information & Communications Technology Planning & Evaluation (IITP), Grant/Award Number: RS-2022-II220961

Abstract

The increasing prevalence of advanced persistent threats and automated cyberattacks highlights the need for proactive cybersecurity strategies. Reinforcement-learning-based autonomous penetration testing has shown promise; however, many existing simulation environments rely on abstract, probability-driven attack models, resulting in limited reproducibility and gaps relative to real-world conditions. This paper presents E-NASim as a reinforcement-learning-based cyberattack simulation framework that explicitly models attack feasibility and system evolution using pre-/post-attack state (PPS) transitions derived from MITRE ATT&CK subtechniques. In E-NASim, attack execution conditions and state transitions are deterministically governed by PPS definitions, whereas reinforcement learning is used to learn effective multistage attack sequencing under fixed constraints. Experimental results demonstrate that E-NASim enables agents to learn coherent and executable multistage attack strategies in complex network environments, providing a practical foundation for automated penetration testing and security evaluation.

KEYWORDS

automated penetration testing, cyberattack simulation, Markov decision process (MDP), MITRE ATT&CK, pre-/post-state modeling, reinforcement learning

1 | INTRODUCTION

Modern cyber warfare is evolving into a sophisticated and stealthy threat landscape combining AI-driven advanced persistent threats with large-scale automated attacks. Traditional defense strategies focus on post-incident measures such as endpoint detection and response and security information and event management. However, these approaches are often vulnerable to evasion techniques and struggle to keep pace with rapid attacks. Consequently, proactive defense strategies emphasizing vulnerability identification and preemptive testing, including breach and attack simulation and penetration testing (PT), are emerging as new standards for cybersecurity [1].

Despite their potential, proactive defense mechanisms remain heavily reliant on highly skilled security professionals (for example, white-hat hackers). Conducting continuous domain-wide testing has inherent limitations. According to MarketsandMarkets, the global PT market

is expected to reach USD 1.7 billion by 2024 and USD 3.9 billion by 2029, with a compound annual growth rate (CAGR) of 17.1%. However, a global shortfall of approximately 3.5 million unfilled cybersecurity jobs is projected by 2025 [2, 3]. This explosive demand, coupled with a shortage of experts, underscores the urgent need for automation and intelligent PT.

To address these challenges, many governments have established cyber ranges as virtualized environments for training and evaluating cyber capabilities. Examples include the US DARPA's NCR, Israel's CyberGymIEC, and the EU's Airbus Cyber Range. These platforms offer domain-specific, scenario-based environments that simulate realistic systems to enhance operational readiness [4–6]. However, the development of realistic and diverse attack scenarios is a critical factor in such environments. Currently, most scenarios rely on expert knowledge or historical incidents and are constructed manually, limiting their ability to reflect modern attack techniques and provide sufficient scenario variability [7]. This shortcoming restricts the effectiveness of training and hinders preparedness for complex real-world threats.

In this context, agent-based PT using reinforcement learning (RL) has gained increasing interest because it enables automated attack scenario generation and optimal path finding from an adversary's perspective. Representative frameworks such as NASim [8] and CyberBattleSim [9] model the PT task as a Markov decision process (MDP) and allow RL agents to learn optimal strategies. However, these simulation environments often over-abstract state and action spaces for the sake of learning efficiency, resulting in a significant reality gap. Specifically, unlike real-world environments, where attack success is conditional on system context (for example, host privilege levels and vulnerabilities), many existing platforms rely on probabilistic models, reducing their reliability and applicability in practice [10–12]. Probability-driven attack success modeling obscures the explicit system conditions required for attack execution, leading RL agents to acquire strategies that may be infeasible or invalid in real-world environments.

This paper proposes a novel RL-based cyberattack simulation framework called E-NASim, which is designed to bridge the reality gap and improve the strategic validity of agent-learned attack policies. This framework incorporates the explicit modeling of pre-/post-attack states (PPSs) and condition-aware reward structures to enable realistic training. Specifically, E-NASim models cyberattacks as condition-driven state transitions by explicitly defining the required pre-state conditions before an attack and the resulting post-state changes after its execution. Rather than relying on stochastic success probabilities, this approach enables RL agents to reason based on concrete system constraints. Within this framework, the role of an RL agent is to learn valid multi-stage attack sequences under PPS-imposed feasibility constraints, focusing on optimal sequencing and decision-making within a realistic attack space, rather than unconstrained exploration.

Our key contributions can be summarized as follows:

- PPS modeling for ATT&CK subtechniques: We structure attack actions at the MITRE ATT&CK subtechnique level and define explicit PPS conditions for each action. This approach enables the realistic modeling of attacker behavior and system changes, allowing RL agents to learn strategies grounded in environmental constraints.
- PPS-based MDP environment formalization: Using the PPS model, we define attack success conditions and the resulting state transitions within a formal MDP, thereby enhancing the reproducibility of our experiments and supporting reliable policy learning by reflecting real-world dependencies.
- Reward function design reflecting action cost and impact: We design a reward structure that considers the strategic impact and execution cost of each attack action, guiding agents toward efficient and goal-oriented attack paths beyond simple goal-reaching strategies.

Unlike existing NASim-based studies, this study did not aim to demonstrate absolute performance superiority. Instead, our experimental evaluation focused on validating whether PPS-driven modeling enables RL agents to learn coherent and executable attack sequences that respect explicit system constraints.

The remainder of this paper is organized as follows. Section 2 reviews the technical background and related work on PT, MDP modeling, and the MITRE ATT&CK framework. Section 3 presents the E-NASim framework and its architecture, including the detailed modeling of attack state transitions. Section 4 describes our experimental setup and evaluates the learning performance of RL agents in multistage scenarios. Finally, Section 5 concludes this paper and discusses future directions.

2 | BACKGROUND AND RELATED WORK

This section presents the theoretical foundation and technical background of the proposed E-NASim framework, followed by an overview of relevant prior work. We first introduce the concepts and automation of PT and then describe how such tasks can be modeled using the MDP.

We also examine the MITRE ATT&CK framework and MITRE CALDERA for the standardized representation and emulation of attacker behavior. Finally, we compare existing RL-based cyberattack simulators and identify the key gaps that our study aims to address.

2.1 | PT and the need for automation

PT is a security assessment method that identifies and evaluates vulnerabilities in an organization's systems, networks, or applications from an attacker's perspective. PT aims to validate the feasibility of real-world cyberattacks and is typically conducted by authorized security experts executing attack scenarios in controlled environments [13].

Technically, PT can be classified into network- and host-based approaches. Network-based testing involves port scanning to detect open services and identify associated vulnerabilities [14], whereas host-based testing inspects system misconfigurations such as insecure file permissions or registry settings [15].

Recent trends have emphasized automation in PT using various tools integrating exploitation functions connected to public vulnerability databases (DBs). These tools primarily target web servers and applications, significantly improving testing efficiency [16, 17]. In particular, various techniques such as attack graphs [18] and rule-based algorithms [19] have been widely applied to the construction of automated testing frameworks.

However, traditional automation approaches are often limited to static or single-host environments, lacking support for dynamic interactions or network-wide evaluations. Additionally, most PT is conducted in the form of a one-time assessment, the validity of which rapidly diminishes as the system configuration changes [20]. These limitations hinder comprehensive scenario-based security validation.

AI-based PT has emerged as a promising approach to addressing these issues. In particular, RL enables continuous, interactive testing by training agents to learn effective attack strategies through trial and error [21, 22]. We extend this paradigm by developing an RL-based penetration agent capable of operating in multihost network environments with state-driven transitions.

2.2 | MDP and reinforcement

RL is a machine learning paradigm in which an agent learns optimal decision-making through interactions with the environment without requiring explicit supervision or labeled data. This problem is typically modeled as an MDP, which is a formal framework for sequential decision-making under uncertainty [23].

An MDP is defined as a tuple $\langle \mathbf{S}, A, P, R, \gamma \rangle$, where $\mathbf{S}$ is the set of states, A is the set of actions, $P(s's, a)$ defines the probability of transitioning to state s' from state s after taking action a, $R(s, a)$ is a reward function specifying the reward received for action a in state s, and $\gamma \in (0, 1)$ is a discount factor that determines the importance of future rewards.

The MDP assumes the Markov property, which states that the next state depends only on the current state and action and not on any prior history. The goal of RL is to learn an optimal policy $\pi(s)$ that selects the best action in each state s to maximize the agent's expected cumulative reward.

During the learning process, an agent observes the current state s_t at each time step t, selects an action a_t according to its policy (that is, $a_t \sim \pi(s_t)$), receives a reward r_t, and transitions to the next state s_{t+1}. By repeating this interaction over several episodes, the agent updates its policy and converges toward an optimal policy π^* that maximizes the expected return. This learning objective can be expressed as follows [24]:

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]. \tag{1}$$

RL algorithms can be broadly categorized into model-based approaches, which utilize explicit environmental models, and model-free approaches, which learn solely through interactions with the environment. Model-free algorithms are further classified into value-based, policy-based, and actor-critic families.

A representative value-based method is Q-learning, which learns the value of each state–action pair (Q-value) through iterative updates based on the Bellman optimality equation [25].

$$Q^{\text{NEW}}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right). \tag{2}$$

However, traditional tabular Q-learning is inefficient for high-dimensional state spaces. To address this issue, deep Q-networks (DQNs) have been introduced, which approximate the Q-function using deep neural networks. Techniques such as experience replay and target networks have been employed to improve stability and convergence [26, 27]. In this study, the proposed E-NASim

model adopts the DQN framework to train cyberattack agents capable of learning effective and strategic penetration paths in complex environments.

2.3 | MDP-based modeling of autonomous PT

To implement RL in autonomous PT, the problem must be formalized as an MDP in which each component maps to an element in the test scenario. The state space includes the observable features of the network, including host configurations, active services, vulnerabilities, and compromise status. These attributes are encoded as structured state vectors. However, as observed by Schwartz and Kurniawati, the size of the state space increases exponentially with the number of hosts, negatively affecting the scalability of training in realistic environments [8].

The action space consists of the cyberattack operations available to the agent, including scanning, exploitation, and privilege escalation. Each action is typically defined as a pair *(host, technique)*, indicating that a specific technique is applied to a particular host. If the scenario includes N hosts and T techniques, then the action space size becomes $N \times T$, offering a uniform representation, regardless of scenario complexity.

The reward function is designed to encourage the efficient compromise of high-value assets while discouraging unnecessary or costly behavior. This function is computed as the total value of compromised hosts minus the total cost of actions performed.

$$\mathcal{R} = \sum_{h \in H} \text{value}(h) - \sum_{a \in \mathcal{A}} \text{cost}(a). \tag{3}$$

Here, H is the set of compromised hosts, $\text{value}(h)$ is the strategic value of host h, A is the set of actions performed, and $\text{cost}(a)$ is the execution cost of action a.

In practice, the transition function P is often unknown or difficult to model explicitly, particularly in dynamic environments. Therefore, model-free RL algorithms have been widely used. Zhou and others demonstrated that algorithms such as DQN, advantage actor–critic (A2C), and proximal policy optimization (PPO) can be effectively applied to autonomous PT [28], allowing agents to learn optimal strategies without requiring a predefined model of the environment.

Such MDP-based formulations enable RL agents to adopt strategic reasoning that aligns with real-world adversarial behavior, serving as a foundational mechanism for simulating realistic threat flows and supporting automated adaptive PT.

2.4 | Standard frameworks for modeling adversarial behavior: MITRE ATT&CK and CALDERA

2.4.1 | MITRE ATT&CK framework

The MITRE ATT&CK (adversarial tactics, techniques, and common knowledge) framework is a comprehensive knowledge base that categorizes adversary behaviors observed in real-world cyberattacks [29]. This framework is widely used by security professionals for threat modeling, detection, and mitigation planning. As of the time of this writing, the MITRE ATT&CK framework includes 14 tactics, 201 techniques, and 424 subtechniques, each of which defines a specific adversarial objective, method of execution, observable detection sources, and potential mitigation strategies [29, 30]. This structured and evolving taxonomy provides a rich foundation for defining granular and meaningful actions within E-NASim.

ATT&CK organizes adversary behaviors into a four-level hierarchy. Tactics represent high-level attacker goals such as initial access or privilege escalation. Techniques define how these goals are achieved (for example, phishing or credential dumping), whereas subtechniques further specify the methods used, including PowerShell (T1059.001) under the Command and Scripting Interpreter (T1059). Procedures document specific real-world examples, including tools and command-line arguments used by threatening actors.

This hierarchical structure enables precise threat detection, the collection of detailed intelligence on adversary behavior, and effective security control mapping. Accordingly, ATT&CK has become the de facto standard for representing attack behavior in both simulation and operational contexts.

2.4.2 | MITRE CALDERA framework

The MITRE CALDERA framework extends ATT&CK by providing an automated platform for adversary emulation. It allows simulated agents to execute real attack commands on live or virtual systems to evaluate detection and response capabilities under realistic conditions [31]. CALDERA defines each attack operation as an ability that maps to a subtechnique and includes attributes such as the required privileges, execution commands, platforms, and executors (e.g., PowerShell or Bash). Abilities are defined in YAML or JSON and can include preconditions, dependencies, and cleanup steps.

Unlike traditional simulators, CALDERA performs behavioral emulations and executes real commands, rather than simulating abstract effects. This approach

significantly improves fidelity and enables the meaningful evaluation of security controls. However, CALDERA is primarily designed for red-teaming and is not structured for RL.

In this study, we consider CALDERA's concept of abilities as a reference but extend this concept by introducing a structured PPS model. Whereas CALDERA focuses on dynamic runtime logic, E-NASim formalizes each subtechnique using static preconditions and postconditions that define system state transitions. These structured transitions are better suited to RL environments, enabling agents to learn from discrete state changes, rather than raw command executions.

This modeling approach enhances the generalizability, reproducibility, and learning efficiency of cyberattack simulations, forming the core foundation of the E-NASim design.

2.5 | Research trends in RL-based cyberattack simulation

Recently, simulation-based environments for automated PT have been actively investigated. One of the most influential platforms is the network attack simulator (NASim) proposed by Schwartz and Kurniawati, which enables RL agents to learn attack policies through interactions with a simulated network environment [8]. NASim defines network topologies, host configurations, services, and vulnerabilities using YAML files and models the entire environment as an MDP. Its modular scenario structure, reproducibility, and compatibility with RL algorithms have made it a foundational tool for follow-up studies.

Despite these advantages, NASim has significant limitations in terms of realism and expressive power. To simplify learning and environmental design, it abstracts key system-level features such as service states, vulnerability types, and privilege levels. Attack outcomes are often modeled probabilistically, rather than through deterministic condition-based transitions. Consequently, NASim-based agents may learn strategies that perform well in simulations but fail to generalize to real-world scenarios, which is a discrepancy known as the "reality gap" [9, 11].

To address these issues, several studies have applied various RL algorithms within the NASim framework. Ghanem and Zhou explored the use of a DQN, PPO, and A2C for autonomous PT [28, 32, 33]. Tran and others proposed a deep hierarchical reinforcement agent to improve learning performance, whereas Zhou introduced the Noisy nets, Dueling network architectures, Soft Q-learning, Prioritized replay, and Intrinsic curiosity-based DQN (NDSPI-DQN) as an action space decomposition method that separates host selection from technique selection for scalability. Sultana compared the learning stabilities of DQN and PPO models across different testing scenarios [34]. Although these studies have expanded the applicability of NASim, they maintained its core abstraction and did not incorporate realistic attack constraints or state-transition dynamics.

Kim and others applied RL to NASim to predict likely future attack actions, focusing on probabilistic attack path forecasting rather than executable attack generation. Although effective for modeling attack tendencies, this approach inherits NASim's stochastic transition model and does not explicitly represent system conditions or deterministic attack feasibility [35].

To increase behavioral realism, some researchers have integrated the MITRE ATT&CK framework into RL environments. Oh and others mapped simplified ATT&CK tactics to phases in RL experiments for algorithm comparison [36]. Nguyen introduced a multilayer action representation to support hierarchical strategy learning in large-scale networks [37]. These approaches improve the structural representation of attack behavior but still lack explicit modeling of preconditions, effects, and system-level transitions, limiting their fidelity and transferability.

Recent studies have attempted to extend NASim to improve its generalizability and realism. For example, Janisch and others proposed NASimEmu, which combines simulations with system-level emulations to evaluate agents in novel scenarios [12]. Reti and others used NASim to evaluate moving-target defense strategies quantitatively, whereas Zhou and others introduced an episodic memory-guided Deep Q-network method (EMG-DQN) as a DQN variant that incorporates episodic memory to improve long-term policy performance [38, 39]. Nguyen and others proposed PenGym as a framework that links NASim with Metasploit for real-environment execution and validation [40]. Huo and others further extended NASim by introducing a probabilistic defender that dynamically patches vulnerabilities, enabling attacker–defender interaction modeling. However, attack execution and state evolution remain probability-driven, without explicit precondition or postcondition modeling of attack actions [41].

Although these approaches enhance realism and extensibility, they generally lack formal definitions of attack conditions, structured state transitions, and context-aware reward functions. In particular, most NASim-based and ATT&CK-integrated approaches rely on implicit or probabilistic transition assumptions, where attack feasibility and effects are not explicitly encoded as state-dependent conditions. This limitation makes it difficult to guarantee that learned attack policies correspond to executable and logically consistent attack sequences in real-world environments.

To overcome these limitations, this paper proposes E-NASim as a framework that maintains the structural flexibility of NASim while introducing MITRE ATT&CK subtechniques as atomic actions and modeling each attack using explicit PPS transitions. In contrast to previous approaches, E-NASim defines attack feasibility based on concrete preconditions and encodes system changes as structured postconditions, thereby enabling condition-based state transitions that more closely reflect real-world behavior. Furthermore, reward functions are designed to reflect the strategic value and cost of actions and guide agents to learn efficient and realistic attack policies.

By combining structured behavior modeling with RL compatibility, E-NASim provides a simulation environment that supports the autonomous generation of realistic attack scenarios and reduces the gap between training and deployment environments. To clarify how this design choice differentiates E-NASim from existing NASim-based and related frameworks, we provide a structured comparison in the following section.

2.6 | Structured comparison with NASim-based frameworks

To position the proposed framework clearly with respect to previous NASim-based and related studies, we provide a structured comparison of representative simulation- and emulation-based approaches.

Table 1 summarizes the key differences in attack modeling assumptions, state transition mechanisms, and the role of RL. Unlike existing frameworks that rely on probabilistic or implicit transitions, E-NASim explicitly models attack feasibility and effects using PPS definitions. This distinction enables RL agents to learn coherent, executable, and multistage attack strategies under explicit system constraints.

3 | E-NASim FRAMEWORK DESIGN

3.1 | Overview of the framework

Although NASim offers a simplified simulation environment for RL-based PT, its abstraction introduces a reality gap that E-NASim aims to overcome. Specifically, E-NASim introduces a structured simulation environment that explicitly models attack conditions and outcomes using subtechniques from the MITRE ATT&CK framework. E-NASim formalizes each attack with defined PPS conditions, representing the required state before an attack can be executed and the expected state after successful execution. This approach enables agents to consider environmental constraints and learn strategies aligned with realistic system transitions.

E-NASim consists of three core stages, as illustrated in Figure 1. The first stage is training scenario construction, where simulation environments are created using YAML-formatted scenario files. Each scenario includes host configurations, asset information, and a set of attack techniques, each of which is defined with corresponding PPS models derived from MITRE ATT&CK subtechniques. These structured definitions allow for the automated generation of high-fidelity, constraint-aware simulation environments.

The second stage is PT MDP modeling, where the scenario is translated into an MDP. The state and action spaces are constructed based on PPS specifications. State transitions are dynamically determined by evaluating whether the current environment satisfies the pre-state of an attack and, if so, by applying the post-state to update the system. The reward function is designed to reflect both the strategic value of the compromised states and the cost of actions, supporting the agent in learning efficient and goal-oriented behaviors.

TABLE 1 Structured comparison with NASim-based and related frameworks.

Framework	Attack modeling	State transition	Role of RL
NASim [8]	Probability-based	Implicit	Policy optimization
Kim and others [35]	Probability-based	Implicit	Attack prediction
Reti and others [38]	Probability-based	Implicit	Defense evaluation
Huo and others [41]	Probability-based (with probabilistic defense)	Stochastic (patching-aware)	Policy learning under attacker–defender dynamics
PenGym [40]	Real execution	Script-driven	Policy transfer
E-NASim (proposed)	PPS-based	Explicit (pre/post)	Sequence learning under constraints

Abbreviations: NASim, network attack simulator; PPS, pre-/post-attack state; RL, reinforcement learning.

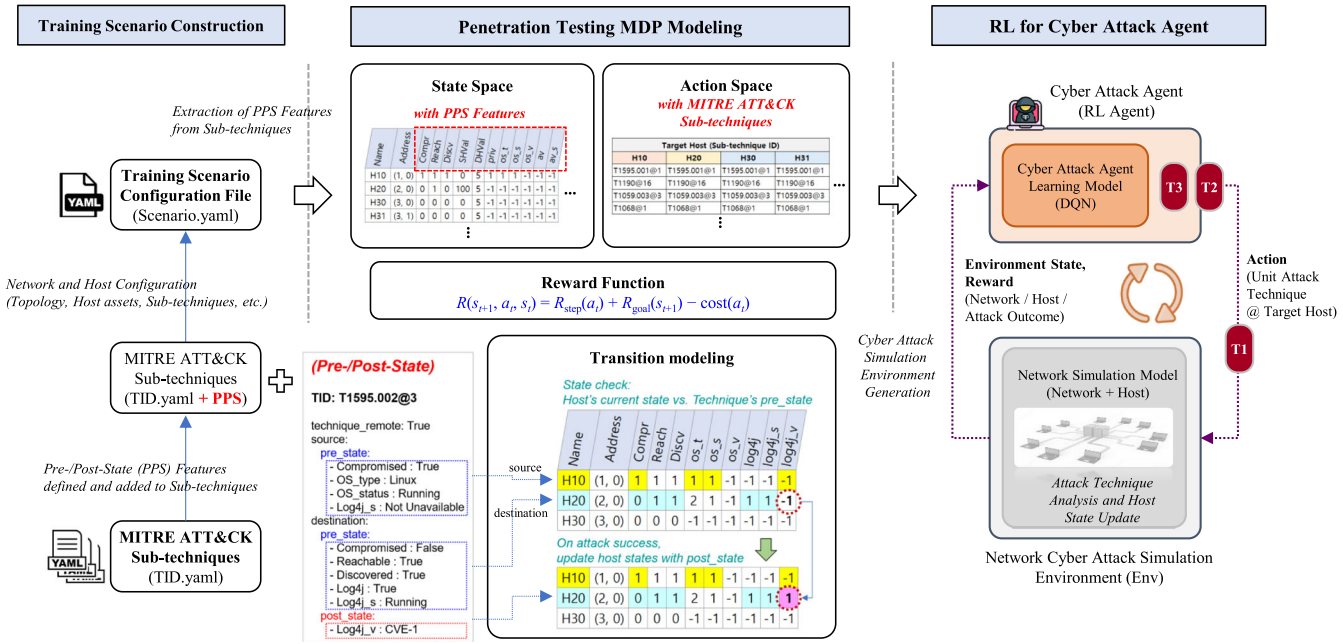


FIGURE 1 Reinforcement learning (RL)-based cyberattack simulation framework (E-NASim).

The final stage is RL for the cyberattack agent, where an RL agent interacts with the environment, observes state vectors, selects attack techniques, and learns from the resulting rewards and transitions. This interaction loop continues until the agent converges toward a policy that can autonomously generate strategic attack sequences under complex constraints.

By modeling attacks as conditional state transitions and encoding environmental logic into both the reward function and MDP structure, E-NASim enables the training of agents that can adapt to realistic system conditions. This framework supports strategic decision-making from the adversary's perspective, rather than relying on random or brute-force action sequences. Consequently, E-NASim provides a simulation platform that balances abstraction with realism, making it suitable for practical applications in PT automation, threat modeling, and security evaluation.

3.2 | Training scenario construction based on PPS modeling

To support realistic and constraint-aware strategy learning, E-NASim defines training scenarios using a structured PPS model. Each attack is modeled as a subtechnique derived from the MITRE ATT&CK framework and is explicitly annotated with a pre-state, describing the environmental conditions required for execution, and a post-state, defining the expected system state after successful execution. These definitions are applied at the level of individual hosts and assets using attribute-based representations.

The scenarios are described in the YAML format, including details such as network topology, host configuration, service states, known vulnerabilities, and sensitive assets. By embedding the PPS logic into each attack action, the environment enforces logical consistency, ensuring that only feasible actions can be executed. This structure enables the agent to learn meaningful, goal-directed behavior, rather than relying on brute-force exploration. Furthermore, this structured modeling approach enhances the extensibility and realism of the simulation environment, making it applicable to a wide range of network configurations and attack paths.

The following subsections detail how attack actions are defined using MITRE subtechniques, how their associated PPS conditions are encoded, and how they are integrated into the broader scenario configuration.

3.2.1 | Defining action units based on ATT&CK subtechniques

In E-NASim, each action available to the RL agent is defined at the level of a subtechnique from the MITRE ATT&CK framework. This design preserves the semantic meaning of attack behaviors while enabling RL agents to make decisions aligned with realistic adversarial strategies. By adopting ATT&CK's structured taxonomy, each action in the simulation corresponds to a concrete and operationally meaningful attack step.

The proposed framework references the ability structure used in the MITRE CALDERA platform, where each ability represents an executable implementation of a subtechnique typically defined by attributes such as commands, privileges, platforms, and executors. Whereas CALDERA focuses on runtime execution logic, E-NASim generalizes this concept by introducing explicit state transitions.

Rather than modeling actions as command-level operations, E-NASim defines each subtechnique in terms of PPSs. The pre-state describes the required environmental conditions under which an action is applicable, including host accessibility, vulnerability, presence, and privilege level. The post-state defines the system changes resulting from successful execution, including privilege escalation, lateral movement capability, or asset compromise.

The PPS model serves multiple purposes, enabling the RL agent to learn about environment-aware transitions, thereby supporting more stable and strategic policy learning. It also ensures logical consistency within simulations by allowing attack actions only when the defined preconditions are satisfied. Furthermore, it allows the agent to explore diverse attack paths by chaining subtechniques that produce different effects toward a common objective.

In summary, whereas E-NASim leverages the standardized structure of ATT&CK subtechniques, it extends them by embedding state transition logic to represent system conditions and effects explicitly. This approach transforms traditional attack descriptions into RL-compatible action units, enabling the more realistic and effective learning of adversarial behaviors in simulated environments.

3.2.2 | Structure and attribute schema of PPS models

E-NASim defines explicit PPS conditions for each subtechnique to enhance the realism and learnability of RL-based cyberattack simulations. These conditions represent the environmental requirements for executing an attack and expected changes in the system state after an attack succeeds. Together, they form a structured PPS model that enables deterministic state transitions and reward assignments based on environmental context.

The structure of each subtechnique is inspired by the ability definition format used in CALDERA, where abilities are described using YAML or JSON and include components such as "pre," dependencies, and executors. The pre field contains optional pre-check commands or initializations; dependencies specify tools or contextual requirements; and the executor defines the execution command and platform. This structure facilitates the operational execution of adversarial actions in real systems.

E-NASim adopts a similar abstraction but reinterprets it in a static, symbolic format to support RL. Rather than relying on runtime conditions and shell-level execution logic, E-NASim represents the PPS using structured attribute–value pairs, allowing this information to be used directly in defining environmental transitions and observations. These structured representations not only simplify the integration of attack conditions into simulations but also allow for the explicit evaluation of action feasibility and impact.

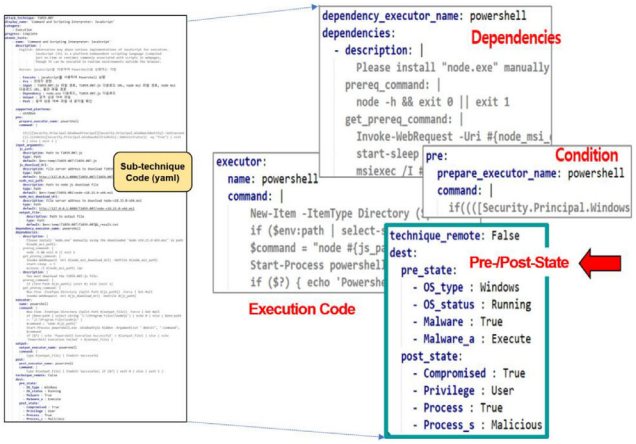
The PPS definitions in E-NASim are currently expert-defined artifacts derived from established adversary knowledge bases, particularly MITRE ATT&CK subtechniques and adversary emulation models such as CALDERA. Therefore, PPS construction is a manual, knowledge-driven process, similar to how attack techniques and prerequisite conditions are curated in real-world PT and red-teaming exercises. We explicitly acknowledge this design choice and discuss it as a limitation and future research direction, including the potential for semiautomated PPS extraction and validation.

Figure 2A presents an example of the YAML definition of a subtechnique in E-NASim, showing how each block is organized and how the PPS structures relate to the learning environment. This format replaces command-oriented action logic with a formal representation of host conditions and changes.

The internal schema of the PPS is organized across multiple layers of granularity. At the host level, covered, compromised, and reachable states are defined using Boolean or categorical values. Host privilege levels (user, admin, and web) are also specified. At the asset level, the attributes include operating systems, services, processes, applications, and files, each with associated status, privilege, access, and vulnerability information.

For example, service types include remote desktop protocol (RDP), virtual network computing (VNC), web, DB, and server message block (SMB), with operational statuses such as running, stopped, unavailable, or infected. Vulnerabilities are mapped to identifiers such as common vulnerabilities and exposures (CVE) numbers. The process and application status may indicate normal or infected execution. Files are characterized by access permissions (read, write, and execute), logical states (created, deleted, and encrypted), and file identifiers.

Figure 2B summarizes this schema in tabular form, organizing attributes into categories such as host, service, file, and privilege. This hierarchical schema enables the fine-grained definition of system states and their changes,



(A)

Category			Attribute	value
Host Status			Discovered, Compromised, Reachable	Boolean
Host Privilege			User, Admin, Web	Integer
Host Assets	OS	Type	Linux, Windows, MacOS	Integer
		Status	Running, Shutdown, Destroyed	Integer
		Vulnerability	CVE-1 ~ CVE-N	Integer
	Service	Type (13)	RDP, VNC, Web ,DB, FireWall, AV, Mail, SSH, FTP, log4j, EternalBlue, Tomcat, SMB	Boolean
		Status	Running, Stop, Unavailable, Infected	Integer
		Vulnerability	CVE-1 ~ CVE-N	Integer
	Process	Type	Process, AuthProcess	Boolean
		Status	Running, Stop, Malicious	Boolean
		Privilege	User, Admin, System	Boolean
	Application	Type	MS-Product, HWP, Chrome, IE	Boolean
		Status	Running, Stop, Unavailable, Infected	Integer
		Vulnerability	CVE-1 ~ CVE-N	Integer
	File	Type (12)	Password, DB_file, Document, NetworkInfo, UserInfo, Tool, Log, Exploit, Malware, OSInfo, Persistence, DefenseEvasion	Boolean
		Status	Init, Created, Deleted, Modified, Discovered, Dumped(Copied), Encrypted, Encoded, Compression, Cracked, Received, Leaked	Integer
		Access	Read, Write, Execute	Integer
		Id	ID-1 ~ ID-N	Integer

(B)

FIGURE 2 Examples of subtechnique definitions and pre-/post-attack state (PPS) schema in a reinforcement learning (RL)-based cyberattack simulator: (A) example YAML definition of an attack subtechnique and (B) attribute schema for representing host and asset states using PPS.

supporting the realistic modeling of attack preconditions and consequences. It also enables the efficient construction of the state space for RL agents, as only relevant attributes referenced by PPS definitions are included in an agent's observations.

By employing a structured and extensible PPS schema, E-NASim enables agents to reason about actions, not just as discrete commands but as transitions within a logically consistent system. This modeling approach provides a foundation for scalable and generalizable learning in realistic cyber threat simulations.

3.2.3 | Enhancing scenario expressiveness through environment configuration

Whereas NASim supports the YAML-based definition of hosts and services, its simplified abstractions limit scenario expressiveness, particularly in modeling multistage attacks involving conditional transitions, privilege dependencies, and sequential tactics.

For example, vulnerabilities in NASim are often represented as simple Boolean flags without modeling their specific types or associated conditions.

Relationships between system components such as privilege boundaries, file locations, and service-to-asset mappings are largely omitted. Furthermore, attack outcomes are typically binary (that is, success or failure) and do not capture how system states evolve as a result of each action. These constraints limit the ability of agents to learn realistic multistage strategies, particularly for internal movements and privilege escalation.

E-NASim addresses these limitations by extending the scenario definition model structurally. Each host is annotated not only with vulnerability presence but also with specific CVE-based attributes that serve as direct references for pre-state conditions. Sensitive data such as password files or configuration entries are represented as assets with explicit properties, allowing simulations to capture lateral movement opportunities, shared credentials, and privileged relationships across hosts.

Attack actions are defined using subtechniques from the MITRE ATT&CK framework, each of which is embedded with structured PPS models. These definitions specify not only whether an attack is possible but also what system state transitions occur upon success. For example, an agent can only perform a lateral movement if it has already acquired credentials from a compromised host, and a successful movement updates the compromise and privilege attributes of the target host accordingly.

This modeling approach allows an RL agent to interact with a simulation environment that reflects the underlying logic and constraints of real attack scenarios. Rather than relying on random exploration, the agent must reason about the system state, environmental conditions, and strategic value of each action. Consequently, the learning process becomes more targeted and efficient.

Ultimately, the extended scenario configuration in E-NASim narrows the gap between abstract simulations and operational reality, supporting the development of agents capable of learning practical attack strategies while maintaining the flexibility and reproducibility of simulation-based experimentation.

3.3 | MDP modeling of the PT environment

To enable strategic attack policy learning through RL, E-NASim formalizes the PT environment as an MDP. The simulation environment is structured into core MDP components, including a network model, state space, action space, transition function, and reward function. Each component is designed to incorporate MITRE-ATT&CK-based attack techniques and PPS logic, allowing condition-driven decision-making and realistic system transitions.

In this section, we explain how E-NASim extends its abstract simulation model to represent realistic and constraint-aware environments. In particular, we describe how hosts and services are modeled, how state and action spaces are constructed from PPS definitions, and how transition and reward functions are configured to reflect environmental feasibility and strategic outcomes.

3.3.1 | Modeling network and host properties

As noted previously, NASim models hosts and services with lightweight abstractions, which E-NASim extends to support detailed system attributes. In NASim, a host, which is referred to as a "machine," is associated with attributes such as subnet membership, running services, and asset value. An RL agent can interact with this environment by navigating the topology and executing actions under firewall constraints. However, to maintain simplicity, NASim limits each host's attributes to basic elements such as service presence and binary vulnerability flags, without modeling deeper relationships or contextual requirements that influence real-world attack feasibility.

This abstraction limits the realism of learned strategies and may lead to policies that are not applicable to real environments. For example, an agent cannot reason about specific vulnerability types, user privilege levels, or asset sensitivity, all of which are critical for adversarial decision-making.

E-NASim addresses this gap by structurally extending the network model to host- and asset-level representations. Each host contains not only basic service and subnet information but also structured properties such as CVE identifiers for vulnerabilities, access-controlled files, installed applications, and active services. These elements are encoded as part of the state vector and used directly to evaluate the PPS conditions of each subtechnique.

Additionally, sensitive data and privileged escalation paths are modeled explicitly. For example, an agent may need to locate and extract a credential file from one host to perform lateral movement to another. Similarly, services such as RDP or SMB are linked to both operational status and exploitability, providing a more accurate simulation of real-world system behavior.

By incorporating this level of detail, E-NASim allows RL agents to reason about the environment in terms of actionable security-relevant conditions. This host model enables the realistic simulation of multistep attack paths, including internal movement and privilege chaining, while preserving compatibility with structured scenario definitions and RL algorithms.

3.3.2 | State space definition based on PPS modeling

In E-NASim, the state space of the RL environment is constructed using the PPS definitions specified for each attack technique. These definitions guide the selection of observable features and determine how the system state is represented and updated. The resulting state space is high-dimensional but structured, enabling agents to reason about attack feasibility and expected outcomes in a realistic manner.

Each host in an environment is represented by a state vector that encodes multiple types of information, including the host's subnet and unique identifier (as a one-hot-encoded address), current security status (for example, discovered, compromised, and reachable), privilege level (for example, user, admin, and web), and the sensitivity or value of associated assets. These core fields are followed by a set of feature values derived from the host's assigned assets and attributes, as referenced in the PPS definitions.

Formally, a host state vector $\mathbf{x}_i$ is defined as

$$\mathbf{x}_i = \Big[\underbrace{\overline{\text{onehot}}(\text{Subnet}_i), \overline{\text{onehot}}(\text{HostNum}_i)}_{\text{HAddr}_i}, \underbrace{c_i, d_i, r_i}_{\text{HStatus}_i}, \underbrace{\text{shval}_i}_{\text{SHVal}_i}, \underbrace{\text{dhval}_i}_{\text{DHVal}_i}, \underbrace{\text{priv}_i}_{\text{HPriv}_i}, \underbrace{f_{i1}, f_{i2}, \ldots, f_{im}}_{\text{HAssets}_i}\Big] \in R^d. \tag{4}$$

The components of the host state vector $\mathbf{x}_i$ are defined as follows.

- HAddr_i represents the one-hot-encoded address of the host, indicating its subnet and ID.
- $c_i, d_i, r_i \in 0,1$ are binary variables indicating whether the host is *compromised*, *discovered*, or *reachable*, respectively.
- $\text{shval}_i, \text{dhval}_i \in R$ denote the sensitivity and detection value associated with the host's assets, respectively.
- priv_i indicates the privilege level of the host, such as *user*, *admin*, or *web*.
- Finally, $f_{i1}, \ldots, f_{im}$ represent the feature values corresponding to the asset attributes derived from the PPS definitions of the attack techniques used in the scenario.

The set of asset features is dynamically derived from the PPS definitions associated with all attack techniques. Specifically, the simulation extracts only those attributes that appear in any PPS model, thereby reducing the dimensionality of the state space and improving learning efficiency. The set of relevant asset features F_{used} is defined as

$$\mathcal{F}_{\text{used}} = \bigcup_{\mathcal{T}_k \in \mathcal{T}_{scenario}} \text{keys}(\mathcal{T}_k), \tag{5}$$

where $\mathcal{T}$ is the set of all subtechniques in the scenario and keys$(\cdot)$ extracts the attribute fields from each PPS definition.

The overall system state $\mathbf{S}$ is formed by concatenating the state vectors of all N hosts in the environment as follows:

$$\mathbf{S} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix} \in R^{N \times d}. \tag{6}$$

In addition to the host states, the agent is provided with a result vector that encodes the outcome of the most recent action. This vector includes information on whether the previous attack was successful and, if not, the reason for failure. Specifically, the result vector $\mathbf{r}_{\text{exec}}$ contains four binary fields: *Success*, *ConnErr* (connection error due to unreachable host), *PermErr* (permission error due to insufficient privilege), and *UndefErr* (undefined or unknown failure). These indicators are followed by padding zeros to match the dimensions of the host state vector.

$$\mathbf{r}_{\text{exec}} = \Big[\text{Success}, \text{ConnErr}, \text{PermErr}, \text{UndefErr}, \underbrace{0, \ldots, 0}_{d-4}\Big] \in R^d. \tag{7}$$

The final extended state matrix observed by the agent is defined as

$$\mathbf{S}' = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \\ \boldsymbol{r}_{\text{exec}} \end{bmatrix} \in R^{(N+1) \times d}. \tag{8}$$

This matrix encapsulates the full environmental state at each time step, including both the system conditions and recent interaction context. This model allows an agent to condition its decisions not only on the system structure but also on past feedback, thereby facilitating more robust and context-aware policy learning.

3.3.3 | Action space definition based on attack techniques

In E-NASim, the action space of the RL environment is constructed based on a set of executable attack

techniques available within the scenario. Each action corresponds to a unit attack, which is defined as the application of a specific attack technique to the host. Formally, the action space consists of all possible pairs (host and technique), where the technique is a subtechnique derived from the MITRE ATT&CK framework.

Let H denote the set of hosts and T denote the set of attack techniques defined in the scenario. Then, the action space A is given by the Cartesian product $A = H \times T$, meaning the agent can, in principle, attempt any technique on a host. This structure assumes that the same set of techniques is uniformly available across all hosts. Although this assumption may not hold in every real-world scenario, it simplifies the implementation and training processes by eliminating the need for per-host action masking.

This uniform design offers practical benefits for RL, including a fixed-dimensional output space in the Q-network or policy model, eliminating the need for dynamic resizing or the filtering of invalid actions during inference. Action feasibility is enforced through the state transition function, which evaluates the PPS-defined preconditions for each technique before execution.

Although the full action space grows linearly with the number of hosts and techniques, its structure remains manageable and well suited to deep RL. Furthermore, the PPS-based transition mechanism ensures that the agent receives meaningful feedback only when actions are contextually appropriate, allowing it to learn action-selection policies that can be generalized across different network structures and scenarios.

This design of the action space, which revolves around host-technique pairs, supports the flexible but consistent simulation of adversarial decision-making, enabling agents to explore diverse attack paths and develop realistic intrusion strategies.

3.3.4 | State transition function based on PPS conditions

E-NASim defines environmental dynamics using a state transition function that explicitly models the effects of attack actions using the PPS structure associated with each technique. This approach formalizes how the system state evolves in response to agent actions, ensuring that state transitions are logically consistent and aligned with real-world attack semantics.

The transition function is defined as

$$s_{t+1} = f(s_t, a_t), \tag{9}$$

where s_t is the current state and a_t is the selected action. This function evaluates whether the pre-state conditions of the selected attack technique are satisfied in the current state. If the conditions are met, then the corresponding post-state is applied to update the system; otherwise, no change occurs, and a failure signal is returned to the agent.

Each attack technique is classified as either local or remote, and the transition logic depends on this classification. During a local attack, an action is evaluated only with respect to a single target host. The pre-state is checked against the current attributes of the host, and if the attack conditions are satisfied, then the post-state is applied to the same host to reflect the result of the attack.

In contrast, a remote attack involves two hosts: the source (initiating host) and the destination (target host). In this case, the transition function evaluates whether both the source and destination hosts meet the pre-state conditions defined in the technique. If so, the post-state is applied independently to both hosts. This dual-condition evaluation supports the realistic modeling of multihost operations such as lateral movement or remote code execution (RCE).

The attack type (that is, local or remote) is specified by the metadata field technique_remote (true/false) within each technique definition. This field informs the environment regarding how to interpret the PPS conditions and which hosts to evaluate or update.

Figure 3 illustrates this distinction, showing how local and remote attacks are processed differently. For local attacks, only the attributes of the target host are involved, whereas remote attacks require condition evaluation and state updates across two hosts.

By structuring state transitions around PPS logic, E-NASim ensures that attack feasibility and effects are determined by the environment state, rather than by fixed probabilities. This approach enables an RL agent to develop policies that are sensitive to realistic system conditions and learn meaningful cause-and-effect relationships. The result is a more stable learning process and a simulation environment that better reflects the logical flow of a real-world cyberattack.

3.3.5 | Reward function design for strategic penetration

The reward function in E-NASim is designed to guide RL agents toward learning strategic and efficient penetration behaviors, rather than merely executing arbitrary or greedy actions. It incorporates not only the success of an attack but also the strategic importance of the outcome and the operational cost associated with the action, enabling agents to develop attack paths that are both effective and resource-aware.

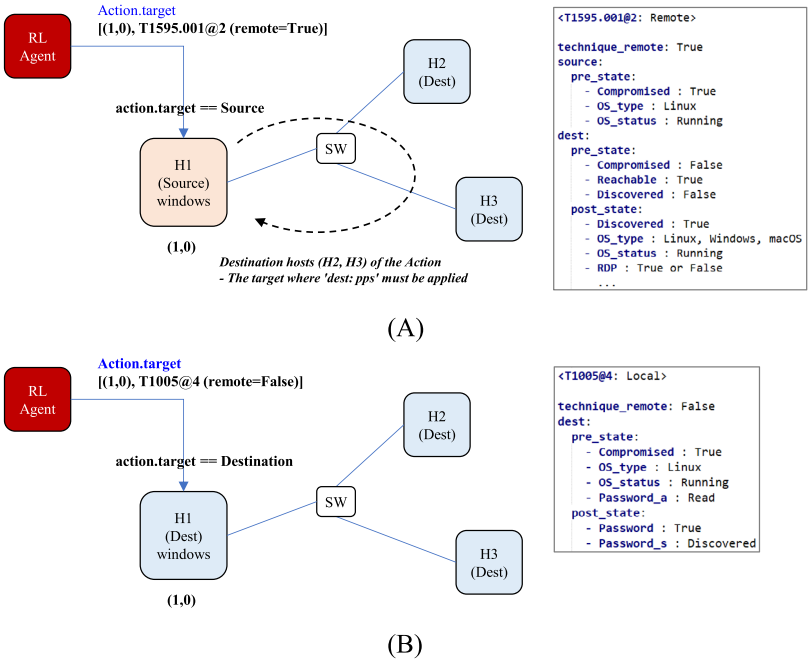


FIGURE 3 Pre-/post-attack state (PPS)-based modeling of local and remote attack techniques in a reinforcement learning (RL)-based cyberattack simulator: (A) PPS evaluation and update flow for remote attacks involving source and target hosts and (B) PPS evaluation and update flow for local attacks on a single host.

The total reward received by the agent in each step consists of three elements:

- a base reward according to the type of action $R_{\text{step}}(a_t)$,
- a goal-oriented reward reflecting successful state transitions $R_{\text{goal}}(s_{t+1})$, and
- a cost penalty associated with resource consumption or risk $\text{cost}(a_t)$.

An action-based reward is a static value assigned to each attack technique based on strategic significance. For example, reconnaissance actions such as scanning or information gathering may yield low rewards (for example, +2), whereas high-impact techniques such as credential theft or RCE may offer higher rewards (for example, +5 to +13). These values are configured manually to encourage the agent to prioritize impactful behaviors.

A goal-based reward is granted when an action leads to a meaningful state transition, such as a host becoming compromised, a privilege level being elevated, or a sensitive file being acquired. These outcomes are determined by comparing the system states before and after the attack based on the post-state definition of the executed technique. For example, compromising a host may yield +12 points, achieving privilege escalation may yield +7 points, and accessing a sensitive asset may yield +13 points.

The action cost represents resource consumption, execution risk, or operational time and is assigned a positive value that is subtracted from the total reward. For example, scanning may incur a cost of one point, whereas more intrusive actions such as privilege escalation or credential theft may cost three to five points. These cost values are manually assigned, reflecting the relative difficulty or risk of each technique.

The total reward R_t at time step t is computed as

$$R(s_{t+1}, a_t, s_t) = R_{\text{step}}(a_t) + R_{\text{goal}}(s_{t+1}) - \text{cost}(a_t). \quad (10)$$

This reward structure encourages an agent to maximize the impact of its actions while minimizing unnecessary costs. By balancing success-driven incentives and penalty terms, the agent is guided to explore not only feasible actions but also strategically valuable actions that align with adversarial objectives in real-world cyber operations.

Such a design supports the emergence of efficient, goal-directed policies and increases the interpretability and realism of learned attack behaviors. This design also ensures that agents learn not only how to succeed but also how to succeed efficiently under realistic operational constraints.

3.4 | Design of the DQN-based learning agent

To enable RL agents to acquire strategic attack behaviors, E-NASim adopts a DQN framework. An agent observes the current state of the environment, selects an attack action from the defined action space, and updates its policy through trial-and-error interactions based on received rewards and state transitions.

The input for the agent consists of the extended state matrix defined in Section 3.3.2. This matrix includes the state vectors of all hosts, as well as a result vector

summarizing the outcome of the agent's previous action. Each host state vector encodes critical attributes such as discovery status, compromise state, privilege level, and selected asset properties derived from PPS definitions. This structured input enables agents to make decisions based on the security-relevant context of the network environment.

Figure 4 illustrates the overall architecture of the DQN-based learning agent. The agent receives the current state s_t and computes a Q-value $Q(s_t, a_t; \theta)$ for each action a_t in the action space $A = H \times T$, where H is the set of hosts and T is the set of available techniques. The Q-values represent the expected cumulative rewards for each action starting from the current state. The Q-function is approximated using a deep neural network, typically implemented as a multilayer perceptron (MLP), with an output dimension equal to the total number of possible (host, technique) action pairs.

In each time step, the agent selects an action using an ε-greedy policy, which balances exploration and exploitation. The selected action is applied to the environment, yielding a reward r_t and the next state s_{t+1}. The tuple (s_t, a_t, r_t, s_{t+1}) is stored in an experience replay buffer, which maintains a fixed-size memory of recent transitions.

During training, the agent samples minibatches of transitions from the replay buffer and updates the Q-network parameters by minimizing the following loss:

$$L = \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta) \right)^2. \quad (11)$$

Here, θ denotes the parameters of the online Q-network and θ^- represents the parameters of the target network, which are periodically updated to stabilize learning. The target value is computed using the target network, whereas the current estimate is produced using the online network. This design prevents feedback loops from arising when both sides of the update equation depend on the same parameters.

The use of experience replay and a separate target network mitigates issues such as catastrophic forgetting and instability during training. These techniques, in combination with the structured inputs and reward design introduced previously, allow DQN agents in E-NASim to learn effective and reusable attack policies that can be generalized across different network environments.

By combining deep Q-learning with PPS-based simulation logic, E-NASim enables agents to learn not only how to compromise systems but also how to do so strategically, based on realistic preconditions, operational constraints, and long-term goals.

4 | EXPERIMENTS

4.1 | Objective and setup

The primary objective of our experiments was to evaluate whether the proposed E-NASim framework can support realistic and condition-aware cyberattack simulations using MITRE-ATT&CK-based techniques and structured state transitions. Specifically, our experiments aimed to determine whether an RL agent can learn effective and strategic attack policies under complex multistage scenarios. Importantly, the objective of our evaluation is not to claim absolute performance superiority over existing simulators but to validate PPS-driven learning behavior by determining whether an RL agent can learn coherent and executable multistage attack sequences under explicit pre-state constraints and post-state transitions.

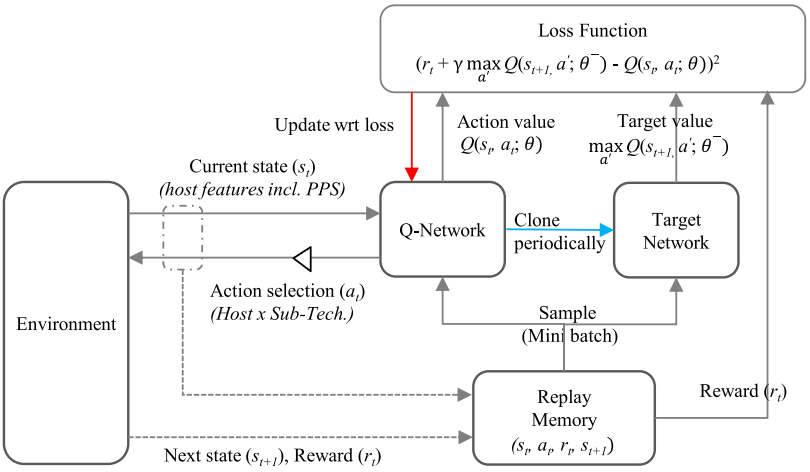


FIGURE 4 Learning architecture of the deep Q-network (DQN)-based cyberattack agent. The agent receives the extended state matrix as an input and estimates Q-values for all actions. Actions are selected using an ε-greedy policy, and the resulting rewards and next states are stored in the experience replay buffer for training. A separate target network is periodically synchronized to stabilize learning.

To this end, two distinct training scenarios, denoted as v1 and v2, were designed, each of which is defined by a set of attack techniques and corresponding PPS models. The RL agent was trained to select actions from these techniques, receiving feedback through a reward function reflecting both strategic value and execution cost. Accordingly, experimental results were interpreted in terms of strategy validity, logical consistency of state transitions, and constraint-aware sequencing, rather than as a direct benchmark of raw learning performance, as in prior studies.

The RL agent was implemented using the DQN algorithm described in Section 3.4. Experiments were conducted on a system equipped with an NVIDIA RTX 2080 Ti GPU, Intel Core i9-11900K CPU, and 128 GB of RAM. All training was performed using PyTorch with the hyperparameters and environmental settings detailed in the following subsections.

4.2 | Scenario design and comparison

To evaluate the policy-learning performance of E-NASim, two training scenarios, v1 and v2, were constructed, each with different levels of structural complexity and attack path diversity. These scenarios differ in terms of network topology, host configuration, service composition, and the number and distribution of attack techniques. Consequently, the structures of the state and action spaces encountered by the agent varied significantly.

Scenario v1 is a small-scale, two-tier network modeled after a simplified real-world breach incident. The attacker begins with an infected endpoint and moves through the relay server toward the internal web and DB servers. This scenario includes two subnets and represents the core stages of multistep attacks, including reconnaissance, exploitation, privilege escalation, information gathering, and data exfiltration, in condensed

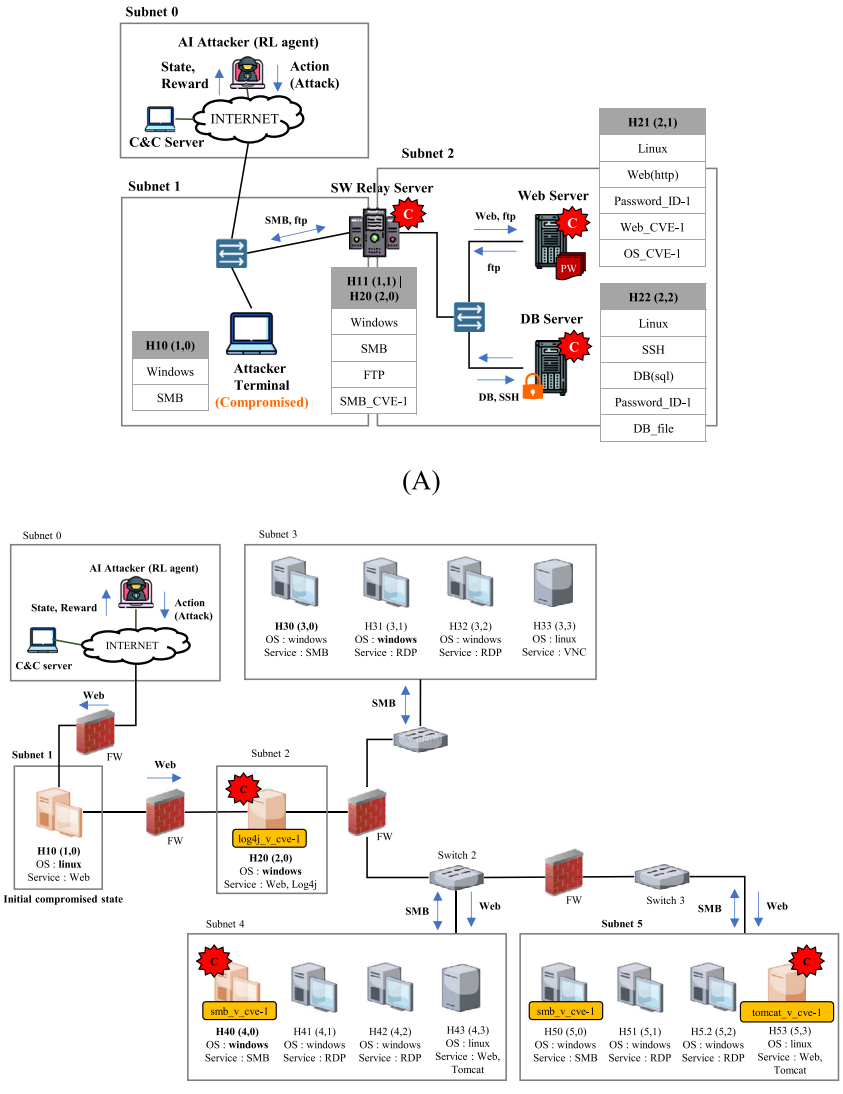


FIGURE 5 Network topologies of training scenarios v1 (A) and v2 (B). (A) A two-tier structure with a single relay path, modeling a simple intrusion flow. (B) A six-subnet architecture with multistage lateral movement, representing complex asset distribution and state transitions.

form. This setup was primarily intended to verify the feasibility of state-transition-based policy learning in a controlled environment.

Scenario v2 simulates a more realistic and complex information technology (IT) infrastructure modeled after a small-to-medium-sized enterprise. The network consists of six interconnected subnets combining external and internal assets. The attacker starts in Subnet 0 and must first gain access to the initial foothold in Subnet 1. Then, the agent must compromise a high-value host in Subnet 2 (H20), move laterally through Subnets 3 and 4, and finally reach a critical Tomcat server in Subnet 5 (H53). Each host is described using multidimensional feature vectors, including operating system, service status, privilege levels, and vulnerability indicators. The agent must condition its actions based on these features to make progress.

Figure 5 presents a visual comparison of the network structures for scenarios v1 and v2, indicating the number of subnets, host relationships, and possible progression paths for the attacker. Whereas scenario v1 offers a narrow deterministic path with limited options, scenario v2 presents a broader and more dynamic environment with multiple attack branches and constraints.

Scenario v2 also includes 40 MITRE-ATT&CK-based techniques, covering diverse tactics such as reconnaissance, initial access, privilege escalation, lateral movement, and exfiltration. Owing to this expanded scope, the complexity of both the state and action spaces is substantially greater than that in scenario v1. Additionally, every attack in this scenario is governed by PPS-based execution conditions that require the agent to learn context-sensitive decision-making strategies.

Although both scenarios utilize structured state transitions based on the PPS model, scenario v2 introduces greater environmental variability and stricter constraints, serving as a more rigorous benchmark for evaluating the generalization capability and realism of learned attack policies.

4.3 | State transition modeling of attack techniques

In E-NASim, each attack technique is modeled as a structured state transition function based on MITRE ATT&CK subtechniques. Every attack includes information such as the execution type (local or remote), preconditions (pre-state), effects (post-state), source and target nodes, and whether the action is classified as remote. These elements allow deterministic transitions between states, enabling RL agents to develop policies that reflect the causal dynamics of real-world cyberattacks.

Each technique can be executed only when its pre-state conditions are satisfied in the current environment. Once an attack is successfully executed, the post-state is applied, resulting in a state change that reflects the intended impact of the attack. This structure establishes a clear map between the agent's actions and the corresponding environmental outcomes, thereby supporting meaningful policy learning.

For example, the agent may begin by scanning the web server in Subnet 1 (H10) using T1595.001@1 to enumerate the Internet Protocol (IP) range and then search for a log4j vulnerability in Subnet 2 (H20) using T1595.002@3. If the required vulnerability is present and the host is reachable, then the agent can launch an RCE attack using T1190@5, resulting in the target host being marked as *compromised* = *true* and *privilege* = *web*.

Table 2 summarizes the three representative subtechniques from scenario v2, including the pre-state conditions, post-state effects, source-to-destination mapping, and whether the action is remote. This structure allows the agent to infer how a given technique affects the environment and when it is appropriate for application.

Each attack is implemented as a discrete transition function, and state transitions are triggered only when a defined pre-state is satisfied. For example, the *compromised* status of host H20 must be set to *true* before lateral movement to H40 can be initiated. This

TABLE 2 State transition examples of three representative attack techniques in scenario v2.

Technique ID	Description	Pre-state conditions	Post-state effects	Source → Destination	Remote
T1190@5 (Exploit Log4j)	Exploit log4j vulnerability on web server	log4j_v = cve-1, reachable = true	compromised = true, privilege = web	H10 → H20	Yes
T1190@3 (Exploit SMB)	Exploit internal SMB vulnerability	smb_v = cve-1, cred_s = discovered	compromised = true, privilege = user	H20 → H40	Yes
T1190@16 (Exploit Tomcat)	RCE on final Tomcat server	tomcat_v = cve-1, reachable = true	compromised = true, privilege = web	H40 → H53	Yes

Abbreviations: RCE, remote code execution; SMB, server message block.

structure allows the agent to discover and sequence attack techniques in a logically consistent manner, mirroring how real attackers operate across the multiple phases of a campaign.

By explicitly modeling both the conditions and effects of attacks, E-NASim enables RL agents to learn not only which actions lead to high rewards but also why and under what circumstances they are effective. This modeling approach strengthens policy interpretability and improves generalization to unseen environments.

TABLE 3 DQN training hyperparameters for scenarios v1 and v2.

DQN hyperparameters	Value (scenario v1)	Value (scenario v2)
Max steps per episode	1000	2000
Discount factor	0.9	0.85
Initial epsilon value	1.0	1.0
Final epsilon value	0.05	0.05
Exploration steps	50 000	100 000
Learning rate	0.001	0.001
Batch size	256	384
Hidden layer size	128	128
Replay memory size	100 000	100 000
Target network update frequency	1000	3000
Training steps	100 000	300 000

Abbreviation: DQN, deep Q-network.

4.4 | Learning results of a DQN-based attack agent

This section analyzes the performance of a DQN-based RL agent when trained in E-NASim using two defined scenarios (v1 and v2). The agent was trained to discover effective attack strategies based on PPS logic and the reward structure described in Section 3.3.5. These experiments assessed whether the agent could learn to generate coherent goal-oriented attack sequences under different environmental complexities.

Table 3 summarizes the hyperparameters used in each scenario. Scenario v2 involves a larger and more complex environment, requiring more training steps, extended exploration, and a higher batch size for stable convergence.

Learning performance was evaluated using two metrics, namely, the average number of steps per episode and episode return, as shown in Figure 6. In scenario v1, the agent quickly converged within approximately 1000 episodes, with the average episode length stabilizing at approximately 17 steps (Figure 6A) and the return reaching approximately 112 (Figure 6B). In scenario v2, which presents a larger and more complex topology, convergence

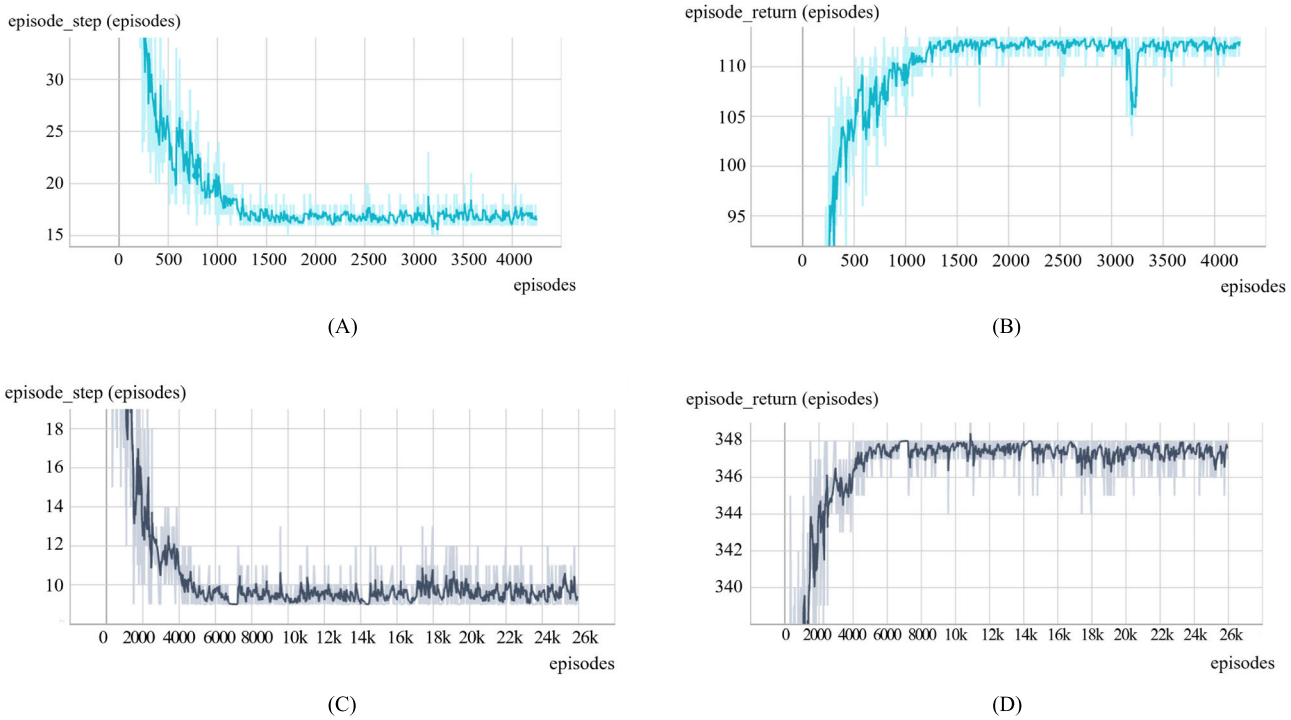


FIGURE 6 (A–D) Comparison of deep Q-network (DQN) agent learning performance in scenarios v1 and v2.

TABLE 4 Summary of the attack scenario generated by the DQN agent (based on scenario v2).

Step	Tactic	Technique	Description
1	Scanning	T1595.001@1	Perform IP range scan
2	Discovery	T1595.002@3	Identify log4j vulnerability
3	Initial access	T1190@5	Exploit log4j RCE to compromise internal host
4	Movement	T1190@3	Move laterally via SMB exploit
5	Discovery	T1595.002@6	Detect Tomcat vulnerability
6	Final access	T1190@16	Execute remote code on final Tomcat server

Abbreviations: DQN, deep Q-network; IP, Internet Protocol; RCE, remote code execution; SMB, server message block.

requires more training. The episode length stabilized at approximately 10 steps after 4000 episodes (Figure 6C), whereas the return converged to 340–350 (Figure 6D). These results demonstrate the ability of the agent to learn multistage attacks by navigating complex environments and satisfying the required pre-state conditions.

These results demonstrate that the DQN-based agent effectively learned the state transition structure and reward design embedded in E-NASim. Rather than relying on random exploration, the agent developed goal-directed policies reflecting strategic reasoning. In particular, even in complex environments such as scenario v2, which require multistage lateral movement and diverse tactics, the agent successfully learned to reproduce expert-level attack sequences through RL.

4.5 | Generation and evaluation of attack scenarios by the trained DQN agent

After training, the DQN-based agent was evaluated by allowing it to generate an attack scenario autonomously starting with the initial state of scenario v2. Each selected action must satisfy the corresponding pre-state conditions for execution. If successful, the post-state is applied to the target host, resulting in a change in the system state. This process continues until the agent reaches the terminal condition or completes a multistage attack sequence.

The learned attack scenario follows a logically coherent sequence that mirrored real-world adversarial tactics. The agent proceeds through the following phases: reconnaissance, initial access, lateral movement, and final access. This sequence reflects a standard kill chain structure and demonstrates the agent's ability to learn and apply a reasonable strategy in a PPS-based environment.

Table 4 summarizes the generated attack scenarios. Each row includes the phase, ATT&CK technique ID, and a brief description of the action. The scenario begins by scanning the external web server (H10), followed by detecting the log4j vulnerability on the internal host H20. The agent then exploits this vulnerability to gain a foothold in Subnet 2. It then performs a lateral movement through the network using SMB exploitation, ultimately reaching the final target, namely, the Tomcat server (H53) in Subnet 5, where it executes an RCE attack.

In this scenario, the agent initiates its actions from the external environment by targeting the web server (H10), discovers a critical vulnerability in H20, and successfully compromises it. Then, the agent identifies SMB- and Tomcat-related vulnerabilities in downstream hosts, performs lateral movement, and finally compromises H53 through RCE.

At each step, the agent evaluates whether the required pre-state conditions are met and only proceeds with actions that are feasible. Upon success, the environment is updated according to the corresponding post-state, thereby enabling the next stage of the attack chain. This conditional progression indicates that the agent internalized a strategy-driven policy, rather than relying on random or exhaustive exploration.

To illustrate how the learned policy operationalizes PPS constraints, we provide a concrete example of a learned attack sequence and its enabling state transitions. Starting from the initial state, the agent first selects a reconnaissance action (for example, T1595.001@1) to discover reachable hosts, which is feasible only when the corresponding subnet-level reachability condition is satisfied. After identifying a valid target and its vulnerability (for example, log4j via T1595.002@3), the agent executes a remote exploitation action (for example, T1190@5) that transitions the target host into $compromised = true$ and $privileged = web$, as specified in the post-state. The policy then performs lateral movement only after the prerequisite conditions are met (for example, discovered credentials or prior compromise), and attempts to execute movement or exploitation without satisfying the pre-state conditions are rejected by the environment, resulting in no state transition or a reduced return. This behavior indicates that the agent learns to avoid invalid action sequences and instead composes executable multistage intrusion paths under explicit system constraints.

The generated scenario demonstrates the agent's ability to synthesize a multistage intrusion path that reflects both tactical planning and environmental awareness. The result highlights the effectiveness of E-NASim for training RL agents capable of producing realistic, coherent attack flows that can be used for simulation-based threat assessment, security validation, or red-team automation.

4.6 | Summary of experimental results and implications

This section summarizes our experimental findings and highlights the practical implications of using the E-NASim framework for RL-based cyberattack simulations.

Overall, the results of our experiments demonstrate that E-NASim enables agents to learn realistic condition-aware attack strategies by explicitly modeling state transitions through the PPS structure. In scenario v1, which featured a compact environment with a simplified topology, the DQN agent converged rapidly. The average number of steps per episode stabilized at approximately 17, and the episode return increased steadily, reaching approximately 112. This scenario validated the feasibility of policy learning using a PPS-based modeling approach.

In contrast, scenario v2 provided a more challenging testbed, incorporating a six-subnet network with multistage lateral movement, privilege escalation, and asset dependencies. Despite this complexity, the agent successfully learned to navigate the environment, achieving convergence after approximately 4000 episodes. The average episode return reached 340–350, and the average number of steps remained at approximately 10. These results confirm that E-NASim supports strategic policy learning, even in high-dimensional environments.

Furthermore, the agent-generated attack scenario reproduced a coherent multistage intrusion flow using real-world vulnerabilities such as log4j, SMB, and Tomcat exploits. The scenario followed a realistic attack chain that mirrored expert-level decision-making, satisfied environmental constraints, and used previously acquired information to inform future steps. This result provides strong evidence that the agent learned not only action–reward associations but also contextual dependencies among subtechniques.

These findings support the conclusion that E-NASim is not merely a behavioral simulator but a structured training platform that enables policy learning under realistic conditions. The framework's explicit modeling of state transitions and condition-based rewards improves the interpretability, transferability, and generalization of learned strategies.

Therefore, E-NASim can serve as a practical foundation for various applications, including automated PT and detection policy validation, security solution benchmarking and evaluation, and simulation-based threat modeling and scenario generation.

5 | CONCLUSIONS AND FUTURE WORK

5.1 | Summary of contributions

This paper proposes E-NASim as a novel RL-based cyberattack simulation framework designed to support realistic and condition-aware PT. Unlike conventional simulators such as NASim, which rely heavily on abstract states and action spaces, E-NASim introduces a structured modeling approach using PPS definitions for each attack technique based on the MITRE ATT&CK framework.

The proposed framework enables the explicit representation of both environmental preconditions and the resulting system changes, bridging the gap between simulated and real-world attack dynamics. E-NASim consists of three main components: (1) PPS-based scenario construction, (2) formalized MDP modeling of the environment, and (3) RL of attack strategies using a DQN agent.

Experimental results reveal that the framework enables agents to learn not only to succeed in achieving compromise goals but also to do so in a strategic and efficient manner. Even in complex scenarios involving multistage lateral movements and layered network topologies, an agent learns to reproduce expert-level attack flows. It should be noted that these results are not intended to demonstrate absolute performance superiority over existing simulators, but rather to validate that PPS-driven modeling enables RL agents to acquire coherent, executable, and logically consistent attack strategies under explicit system constraints.

By supporting deterministic state transitions and goal-oriented policy learning, E-NASim provides a practical foundation for security automation tasks such as PT, threat scenario generation, and defense evaluation. Through this design, the primary contribution of E-NASim is improving the structural realism and interpretability of learned attack policies, rather than optimizing raw learning efficiency or benchmark scores.

5.2 | Future research directions

To enhance the utility and generalizability of E-NASim further, we plan to extend this study in the following directions.

1. Integration of diverse deep RL algorithms: Beyond DQN, we aim to incorporate algorithms such as PPO

and the asynchronous A2C to compare their convergence stability, sample efficiency, and policy diversity in PPS-constrained environments.
2. Evaluation of policy transfer and generalization: We will investigate the ability of learned policies to generalize across different network topologies, host configurations, and asset distributions. This research will include testing whether the same attack strategies can be reused or adapted in scenarios with altered conditions, thereby assessing the transferability and generalization performance of RL-based attack agents.
3. Pre-filtering of infeasible or contextually invalid actions: To improve learning efficiency and sample utilization further, we plan to implement a pre-filtering mechanism that excludes actions deemed invalid based on the current state context. This approach can reduce unnecessary exploration and accelerate policy convergence, particularly in large-scale environments with dense action spaces.

Additionally, future work will explore methods to reduce the manual effort required to construct PPS definitions. Although this study relied on expert-defined PPS models derived from MITRE ATT&CK knowledge, potential extensions include the semiautomated extraction of preconditions and effects from adversary emulation frameworks, threat intelligence reports, or execution traces, as well as the consistency checking and validation of PPS specifications.

These extensions will support the practical adoption of E-NASim as a next-generation platform for intelligent PT and adversarial simulation in cybersecurity research and practice.

ACKNOWLEDGMENTS

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea Government (MSIT) (No. RS-2022-II220961, Development of Core Technology for Self-evolving AI-based Cyber Range).

CONFLICT OF INTEREST STATEMENT

The authors declare that there are no conflicts of interest.

ORCID

Ki Jong Koo https://orcid.org/0009-0005-4333-5294

REFERENCES

1. Gartner, *Hype cycle for security operations* (2023), Accessed Aug. 2025, https://www.gartner.com/en/documents/4547399
2. MarketsandMarkets, *Penetration testing market* (2024), Accessed Aug. 2025, https://www.marketsandmarkets.com/Market-Reports/penetration-testing-market-13422019.html
3. S. Morgan, *Cybersecurity jobs report: 3.5 million unfilled positions in 2025*, Cybersecurity Ventures (2003), Accessed Aug. 2025, https://cybersecurityventures.com/jobs/
4. M. Park, H. Lee, Y. Kim, K. Kim, and D. Shin, *Design and implementation of multi-cyber range for cyber training and testing*, Applied Sciences **12** (2022), no. 24, 12546.
5. CybergymIEC, *Cyber training arena*, Accessed Aug. 2025, https://www. cybergymiec.com/platforms-overview/.
6. Airbus, *CyberRange: advanced simulation and training solution*, Accessed Aug. 2025, https://cyber. airbus.com/en/products/cyberrange.
7. Y. Yang and X. Liu, *Behaviour-diverse automatic penetration testing: a curiosity-driven multi-objective deep reinforcement learning approach*, arXiv preprint (2022), DOI 10.48550/arXiv:2202.10630
8. J. Schwartz and H. Kurniawati, *Autonomous penetration testing using reinforcement learning*, arXiv preprint (2019), DOI 10.48550/arXiv:1905.05965
9. Microsoft Defender Research Team, *Cyberbattlesim*, Accessed May 2025, https://github.com/microsoft/cyberbattlesim
10. F. Caturano, G. Perrone, and S. P. Romano, *Discovering reflected cross-site scripting vulnerabilities using a multiobjective reinforcement learning environment*, Comput. Secur. **103** (2021), 102204.
11. L. Li, J.-P. S. El Rami, A. Taylor, J. H. Rao, and T. Kunz, *Unified emulation-simulation training environment for autonomous cyber agents* (Proc. Machine Learning for Networking, Paris, France), 2022.
12. J. Janisch, T. Pevný, and V. Lisý, *NASimEmu: network attack simulator & emulator for training agents generalizing to novel scenarios* (Proc. European Symposium on Research in Computer Security, The Hague, The Netherlands), 2023, pp. 589–608.
13. P. Vats, M. Mandot, and A. Gosain, *A comprehensive literature review of penetration testing & its applications* (Proc. International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India)), 2020, pp. 674–680.
14. M. Alhamed and M. M. H. Rahman, *A systematic literature review on penetration testing in networks: future research directions*, Applied Sciences **13** (2023), no. 12, 6986.
15. K. Scarfone, M. Souppaya, A. Cody, and Orebaugh, A., *Technical guide to information security testing and assessment*, NIST, Gaithersburg, MD, USA, SP 800-115 (2008), 1–80.
16. G. Yadav, K. Paul, A. Allakany, and K. Okamura, IoT-pen: a penetration testing framework for IoT (Proc. International Conference on Information Networking (ICOIN), Barcelona, Spain), 2020, pp. 196–201.
17. V. Casola, A. D. Benedictis, M. Rak, and U. Villano, *A methodology for automated penetration testing of cloud applications*, Int. J. Grid Utility Comput. **11** (2020), no. 2, 267–277.
18. C. Phillips and L. P. Swiler, *A graph-based system for network-vulnerability analysis* (Proc. Workshop on New Security Paradigms, Charlottesville, VA, USA), 1998, pp. 71–79.
19. J. Zhao, W. Shang, M. Wan, and P. Zeng, *Penetration testing automation assessment method based on rule tree* (Proc. IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), Shenyang, China), 2015, pp. 1829–1833.
20. J. Y. Lee, D. S. Moon, and I. K. Kim, *Technological trends in cyber attack simulations*, Electron. Telecommun. Trends **35** (2020), no. 1, 34–48.

21. A. Chowdhary, D. Huang, J. S. Mahendran, D. Romo, Y. Deng, and A. Sabur, *Autonomous security analysis and penetration testing* (Proc. International Conference on Mobility, Sensing and Networking (MSN), Tokyo, Japan), 2020, pp. 508–515.
22. Z. Hu, R. Beuran, and Y. Tan, Automated penetration testing using deep reinforcement learning (Proc. IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)), 2020, pp. 2–10.
23. R. Bellman, *A Markovian decision process*, J. Math. Mech. **6** (1957), no. 5, 679–684.
24. R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, 2nd ed., The MIT Press, Cambridge, MA, US, 2018, 1–264.
25. C. J. C. H. Watkins and P. Dayan, *Q-learning*, Mach. Learn. **8** (1992), 279–292.
26. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing Atari with deep reinforcement learning*, arXiv preprint (2013), DOI 10.48550/arXiv:1312.5602
27. V. Mnih, K. Kavukvuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaram, D. Wierstra, S. Legg, and D. Hassabis, *Human-level control through deep reinforcement learning*, Nature **518** (2015), no. 7540, 529–533.
28. S. Zhou, J. Liu, D. Hou, X. Zhong, and Y. Zhang, *Autonomous penetration testing based on improved deep Q-network*, Appl. Sci. **11** (2021), no. 19, 8823.
29. The MITRE Corporation, *MITRE ATT&CK®*, Accessed Aug. 2025. https://attack.mitre.org/
30. R. Alford, D. Lawrence, and M. Kouremetis, *CALDERA: a red-blue cyber operations automation platform* (Proc. International Conference on Automated Planning and Scheduling), 2022, pp. 1–2.
31. The MITRE Corporation, *MITRE CALDERA*, Accessed Aug. 2025. https://caldera.mitre.org/
32. M. C. Ghanem and T. M. Chen, *Reinforcement learning for efficient network penetration testing*, Inform **11** (2020), no. 1, 6.
33. 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 (2021), DOI 10.48550/arXiv:2109.06449
34. M. Sultana, A. Taylor, and L. Li, Autonomous network cyber offence strategy through deep reinforcement learning (Proc. Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III), 2021, pp. 490–502.
35. B. Kim, J. Kim, and M. Kim, *A study of reinforcement learning-based cyber attack prediction using network attack simulator (NASim)*, Semicond. Display Technol. **22** (2023), no. 3, 112–118.
36. S. H. Oh, J. Kim, J. H. Nah, and J. Park, *Employing deep reinforcement learning to cyber-attack simulation for enhancing cybersecurity*, Electronics **13** (2024), no. 3, 555.
37. H. V. Nguyen and T. Uehara, *Multilayer action representation based on MITRE ATT&CK for automated penetration testing*, J. Inf. Process. **31** (2023), 562–577.
38. D. Reti, K. Elzer, D. Fraunholz, D. Schneider, and H.-D. Schotten, *Evaluating deception and moving target defense with network attack simulation* (Proc. ACM Workshop on Moving Target Defense, Los Angels, CA, USA), 2022, pp. 45–53.
39. Z. Zhou, T. Zhou, J. Xu, and J. Zhu, *Efficient penetration testing path planning based on reinforcement learning with episodic memory*, Comput. Model. Eng. Sci. **140** (2024), no. 3, 2613–2634.
40. T. Nguyen, Z. Chen, K. Hasegawa, K. Fukushima, and R. Beuran, *PenGym: pentesting training framework for reinforcement learning agents* (Proc. International Conference on Information Systems Security and Privacy, Rome, Italy), 2024, pp. 498–509.
41. Y. Huo, S. You, and M. Jin, *Construction of network pentest-defense adversarial environment based on NASim*, Int. J. Inf. Secur. **24** (2025), no. 166, 1–24.

AUTHOR BIOGRAPHIES

KiJong Koo is a principal researcher at the Electronics and Telecommunications Research Institute (ETRI), Daejeon, Republic of Korea, and is currently working toward a PhD in radio and information communications engineering at Chungnam National University, Daejeon, Republic of Korea. He received his BS and MS degrees in electronics engineering from Chungnam National University, Daejeon, Republic of Korea, in 1999 and 2001, respectively. Since 2000, he has been with the ETRI, where he is currently a principal researcher in the Cyber Security Research Division. His recent research interests include AI-based cybersecurity, autonomous penetration testing, intelligent cybersecurity, and network security.

Daesung Moon received his MS degree in computer engineering from Pusan National University, Busan, Republic of Korea, in 2001 and his PhD degree in computer science from Korea University, Republic of Korea, in 2007. He joined the Electronics and Telecommunications Research Institute (ETRI), Daejeon, Republic of Korea, in 2000, where he is currently a principal researcher. His research interests include network security, data mining, and AI security.

ByungChul Kim received his BS degree in electronics engineering from Seoul National University, Seoul, Republic of Korea, and his MS degree and PhD in electronics engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea, in 1988, 1990, and 1996, respectively. From 1993 to 1999, he worked as a research engineer at Samsung Electronics in

Suwon, Republic of Korea. Since 1999, he has been a professor in the Department of Radio and Information Communications Engineering, Chungnam National University, Daejeon, Republic of Korea. His research interests include computer networks, wireless Internet, sensor networks, and mobile communication.

JaeYong Lee received his BS degree in electronics engineering from Seoul National University, Seoul, Republic of Korea, and his MS degree and PhD in electronics engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea, in 1988, 1990, and 1995, respectively. From 1990 to 1995, he worked as a research engineer at the Digicom Institute of Information and Communications, Seoul, Republic of Korea. Since 1995, he has been a professor in the Department of Radio and Information Communications Engineering, Chungnam National University, Daejeon, Republic of Korea. His research interests include computer networks, wireless Internet, sensor networks, and mobile communication.

How to cite this article: K. J. Koo, D. Moon, B. C. Kim, and J. Y. Lee, *E-NASim: A reinforcement-learning-based cyberattack simulation framework with pre-/post-state modeling based on MITRE ATT&CK*, ETRI Journal (2026), 1–22, DOI 10.4218/etrij.2025-0365