

Adversarial Reinforcement Learning for Offensive and Defensive Agents in a Simulated Zero-Sum Network Environment

Abrar Shahid¹, Ibteeker Mahir Ishum¹, AKM Tahmidul Haque¹,
M Sohel Rahman², A. B. M. Alim Al Islam Razi²

¹ Notre Dame College, Dhaka

² Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology

Abstract

We present a controlled study of two competing reinforcement learning agents in a custom OpenAI Gym-style environment that models offensive brute-force attacks and reactive defenses on a multi-port service. The environment captures realistic trade-offs that model background traffic, brute-force exploits, IP-based evasion, traps, and rate-limiting defenses. Agents are trained using deep Q networks (DQNs) with a zero-sum reward structure. Successful exploits give large terminal rewards, while step actions incur small costs. We evaluated value-based agents in multiple locations, including trap probability, exploit difficulty, and training regimen. The results demonstrate that the observability of the defender and the effectiveness of the trap strongly hinder exploitations. In this scenario, reward shaping and training scheduling are crucial for learning stability. We provide implementation details, reproducible configurations, and guidance for future extensions.

1 Introduction

In recent years, cyberattacks have become increasingly sophisticated that cause widespread damage to critical infrastructure, financial systems, and everyday online services [1]. High-profile incidents such as large-scale ransomware outbreaks and coordinated brute-force attacks demonstrate how traditional rule-based defenses often fail to adapt quickly enough. This motivates the need for autonomous systems that can proactively respond to evolving threats.

Automated decision systems increasingly mediate interactions between attackers and defenders in networked systems [2]. Research on such interactions benefits from controlled simulation environments that allow repeatable experiments and careful measurement of algorithmic behaviors. We introduce a Gym environment focusing on offensive brute-force-type exploitation and on defensive countermeasures that include IP rate-limiting, port rate-limiting, traps, and port closure. [3] [4]

The environment reflects two important properties:

- (1) Normal user traffic produces substantial noise obscuring attacks, and
- (2) Successful exploitation requires sustained request volume, creating a tradeoff for the attacker between persistence and detectability.

1.1 Reinforcement Learning (RL)

Reinforcement Learning (RL) provides a framework for agents to learn optimal strategies through interaction with an environment. At each discrete timestep, an agent observes a state $s \in \mathcal{S}$, selects an action $a \in \mathcal{A}$ according to a policy $\pi(a|s)$, and receives a reward $r \in \mathbb{R}$ that quantifies the immediate outcome, where \mathcal{S} refers to the observation space, \mathcal{A} refers to the action space, and \mathbb{R} refers to the rewards possible under discrete actions. The agent's goal is to learn a policy that maximizes the expected cumulative reward over time. [5] Value-based methods, such as Q-learning [6] and Deep Q-Networks (DQN) [7], estimate the action-value function $Q(s, a)$ to guide decision-making. Exploration strategies like ϵ -greedy are typically used to balance trying new actions with exploiting known good actions.

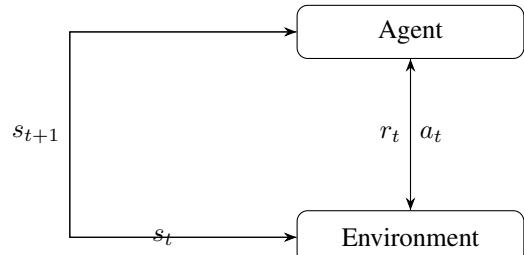


Figure 1: Reinforcement Learning agent workflow in an environment

1.2 Markov Decision Process (MDP)

A Markov Decision Process (MDP) [8] is a mathematical framework commonly used in reinforcement learning to model decision-making problems. An MDP is defined as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where:

- \mathcal{S} is the finite set of states,
- \mathcal{A} is the finite set of actions available in each state,
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the state transition function, and
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the immediate reward function.

In our zero-sum cyber security simulation, the attacker and defender interact sequentially through this MDP. At each timestep t , the attacker performs action $a_t \in \mathcal{A}(s_t)$ on state $s_t \in \mathcal{S}$, producing an intermediate state $s'_t = \psi(s_t, a_t)$ (e.g., a successful scan or exploit attempt). The environment responds with a reward $r_t = \mathcal{R}(s_t, a_t)$ and updates to the next state $s_{t+1} \sim \mathcal{P}(s'_t, s_{t+1})$ (e.g., defender response such as rate-limiting or trap activation). The process repeats until a terminal state $s_T \in \mathcal{S}$ is reached, where the attacker or defender cannot act further.

The goal in this zero-sum environment is to find a policy π for each agent that maximizes its cumulative reward while minimizing the opponent's gain. The state-value function for the attacker, for instance, is defined as

$$V(s_t) = \mathbb{E}[r_t + r_{t+1} + \dots], \quad \dots \dots \dots \quad (1)$$

and the optimal policy can be derived as

$$\pi(s_t) = \arg \max_{a_t \in \mathcal{A}(s_t)} \left(r_t + \sum_{s_{t+1}} \mathcal{P}(s'_t, s_{t+1}) V(s_{t+1}) \right), \quad \dots \dots \dots \quad (2)$$

Here $s'_t = \psi(s_t, a_t)$ represents the intermediate state after the action.

This paper reports a set of experiments that use value-based agents to study learning dynamics, policy stability, and strategic outcomes. We aim to identify the factors that favor defensive or offensive success and to provide methodological recommendations for further work. Our contributions are:

- (1) A detailed environment design that captures brute-force exploits and defender actions;
- (2) An empirical study across multiple ablations; and
- (3) Practical recommendations to improve training stability and interoperability.

2 Methodology

This section describes the environment, the agents, and the experimental protocol. We give precise definitions of actions, observations, rewards, and the exploitation mechanics.

2.1 Environment Overview

The environment simulates a multi-agent zero-sum cyber security scenario, modeling a host with N independent ports. In each simulation there are M IPs among which most are normal users and some are reserved for attacker which are randomized on each episode. At the beginning of each episode, a random subset of ports is designated as *vulnerable*, and each vulnerable port is assigned an exploitation threshold T_p , sampled randomly with a minimum value $T_{\min} = 300$. Exploitation progresses by sending attack-specific requests to the target port from a source IP address. The attacker can change its IP periodically, simulating the use of proxies in real-world attacks, but only after a predefined number of actions.

The environment also generates background normal traffic at each timestep to represent legitimate network activity. Normal traffic is generated from a pool of IP addresses, and the defender must avoid blocking legitimate users while mitigating attacks.

Time evolves in discrete steps, and at each step, only one agent acts: first the attacker, then the defender. The defender is given the option to implement no action in certain steps and rather observe the outcomes. Some actions have instantaneous effects (e.g., scanning a port, setting a trap, or closing a port), while others initiate continuous processes (e.g., an exploit attempt accumulates requests over multiple timesteps). The environment enforces action alternation between agents, but continuous processes such as exploit progression and normal traffic generation proceed independently between actions.

The environment maintains a history of requests, including attacker and normal traffic, to calculate observations for both agents. Observations for the attacker include scanned ports, exploit progress, and current IP status, while defender observations include port request counts, suspicious activity indicators, top IP activity, and current defense configurations. Rewards are structured to enforce a zero-sum dynamic: attacker gains correspond to defender losses and vice versa.

2.2 Action Spaces

In reinforcement learning, the *action space* \mathcal{A} defines the set of all possible actions an agent can take at a given timestep. Each action $a \in \mathcal{A}$ influences the environment state and contributes to the cumulative reward objective. In our zero-sum adversarial setting, both the attacker and defender are restricted to discrete, finite action spaces to ensure tractability and reproducibility.

They are limited to choose each action from the action spaces depending on the scenario.

- **Attacker:** Scan port i for $i \in \{1 \dots N\}$; Exploit port i for $i \in \{1 \dots N\}$; Change IP (after a minimum of 10 actions); Cancel exploit.

The attacker can: Scan ports to detect vulnerabilities, Launch exploits against discovered vulnerable ports, which accumulate over successive timesteps until success, failure, or interruption, Change its IP address to evade detection and bypass rate limits. Cancel ongoing exploits to adjust strategy in response to defender actions.

- **Defender:** Wait(No action); Rate-limit IP q for $q \in \{1, \dots, M\}$; Rate-limit port j for $j \in \{1 \dots N\}$; Set trap on port k for $k \in \{1 \dots N\}$; Close port l for $l \in \{1 \dots N\}$

The defender can: Rate-limit traffic from specific IP addresses or ports, Rate-limit an IP to a number of requests, Deploy traps on ports to penalize attackers, Close ports to completely block access or Simply initiate no action for that step and rather observe the outcomes for further strategies. The ports here are discrete.

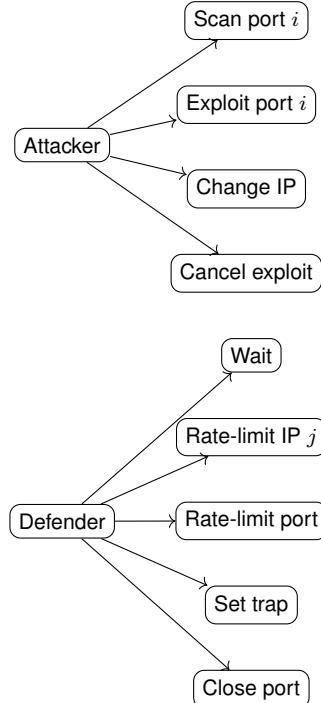


Figure 2: Flowchart of attacker and defender action spaces.

These discrete actions capture standard reconnaissance, exploitation, and defense primitives while keeping the action

dimensions tractable.

2.3 Observation Spaces

The *observation space* \mathcal{O} defines the set of all possible observations an agent can receive from the environment at each timestep. Each observation $o \in \mathcal{O}$ encodes relevant information about the environment state that the agent can use to select actions and maximize its expected cumulative reward. Each continuous observation is discretized into 3 bins, striking a balance between state variability and memory usage. Formally, the state vector for an agent at timestep t is $s_t \in \mathcal{S} \subset \mathbb{R}^{d_{\text{obs}}}$, which is normalized and fed into the Deep Q-Network. This representation allows the agents to learn policies despite partial observability of the environment.

Attacker: The attacker receives a continuous observation vector encoding the results of recent port scans, exploit progress indicators, and a short history of its past actions with their outcomes. Scan results capture both whether a port appears vulnerable and whether defensive anomalies were triggered. An anomaly signal suggests a probabilistic trap by the defender. The attacker also observes its current IP status (e.g., active, or blacklisted).

Defender: The defender receives a continuous observation vector summarizing traffic statistics per port, suspicious activity ratios aggregated over source IPs, and a sliding window of attacker IP histories. In addition, it maintains awareness of the current status of defenses (rate limits, traps, or port closures) and their effectiveness in past interactions. This reflects the log-based perspective of a real-world security operations center, where defenders rely on accumulated traffic data and historical patterns to anticipate intrusions using global tools like SIEM and Elastic.

2.4 Exploration Strategy

The exploration strategy for both the attacker and defender agents is based on the epsilon-greedy approach [9]. In the early training phases, the agents are encouraged to explore the environment by selecting random actions with high probability. As training progresses, the exploration rate decays, and the agents gradually shift towards exploitation of learned strategies.

$$a_t = \begin{cases} \arg \max Q(s_t, a) & \text{with probability } 1 - \varepsilon, \\ \text{random action from } \mathcal{A} & \text{with probability } \varepsilon \end{cases} \quad \dots \dots \dots \quad (3)$$

Here a_t is the action taken, \mathcal{A} is the action space and $\arg \max Q(s_t, a)$ is the best learned action.

The table below outlines the exploration parameters for both agents:

Table 1: Exploration Parameters for the Attacker Agent

Parameter	Value
Learning Rate (α)	0.001
Discount Factor (γ)	0.95
Initial Exploration (ϵ)	1.0
Epsilon Decay Rate	0.995
Minimum Epsilon (ϵ_{\min})	0.05

Table 2: Exploration Parameters for the Defender Agent

Parameter	Value
Learning Rate (α)	0.002
Discount Factor (γ)	0.90
Initial Exploration (ϵ)	1.0
Epsilon Decay Rate	0.99
Minimum Epsilon (ϵ_{\min})	0.05
Batch Size	512

These parameters ensure both agents to learn efficiently from the environment. The learning rate (α) determines how quickly the agents update their policies, while the discount factor (γ) ensures the balance of long-term rewards with short-term gains. [10]. For the attacker, a rate of 0.001 facilitates faster convergence to an optimal policy, while for the defender, a slightly higher rate of 0.002 enables quicker adaptation to the attacker's strategies. Both agents use a high discount factor to balance immediate and future rewards. Initially, both agents explore the environment aggressively, but over time the exploration rate decays, helping the agents shift from exploration to exploitation as they learn optimal strategies. For the attacker, the decay rate is 0.995, meaning it reduces exploration slightly slower than the defender, whose decay rate is 0.99. These rates enable the agents to explore aggressively in the early stages, while reducing exploration as they approach optimal policies. The epsilon decay along each episode is as follows:

$$\begin{aligned} \epsilon(t) &= \max(\epsilon_{\min}, \epsilon_{\text{initial}} \cdot (\text{decay_rate})^t), \\ \text{When } \epsilon(t) = \epsilon_{\min}, \quad t &= \frac{\ln\left(\frac{\epsilon_{\min}}{\epsilon_{\text{initial}}}\right)}{\ln(\text{decay_rate})} \end{aligned} \quad \dots \quad (4)$$

Nearly after t episodes with progressive epsilon decay, the defensive or offensive algorithm follows the best strategy. The epsilon hits the minimum value resulting in the algorithm to always choose the action $\arg \max Q(s_t, a)$ with highest probability.

2.5 Episode Mechanism

2.5.1 Exploitation Mechanics

Exploitation is modeled as a threshold accumulation of attack requests per (IP, port) pair. Each exploit attempt increments a counter $c_{ip,p}$. An exploit succeeds if $c_{ip,p} \geq T_p$ for a vulnerable port p . The threshold T_p is sampled at episode start with $T_p \geq T_{\min}$. The attacker may change its source IP after at least 10 requests from the current IP; changing the IP resets the attacker-specific counter for the next IP. So, The attacker needs to be careful regarding his choice for IP change since changing the IP resets the counter while as detection of multiple request from same IP might result in rate limiting

2.5.2 Defense Mechanics

A trap placed on a port has a probability P_{detect} of being indicated as an anomaly when scanned, and penalizes attackers who reach the trap while exploiting. Traps are intended to create a strategic deterrent and increase defender rewards when triggered. Rate limiting a port to request capacity hampers benign user requests as well. Closing a port implies shutting down a service to all users and incurring a great loss to defend the exploit.

3 Rewards Design

Crucially, the interaction between attacker and defender is inherently a *zero-sum environment*: every successful exploitation by the attacker represents a direct loss for the defender, while effective defense simultaneously denies the attacker's reward. [11] This adversarial framing makes RL particularly well-suited for studying cyber conflict, as it mirrors the strategic interplay where one side's gain is exactly the other's harm. [12].

$$r_{\text{attacker}} = -r_{\text{defender}}.$$

Table 3: Attacker Reward Design

Action	Reward
Successful Exploit	+100
Trap Hit	-80
Scan Cost	-0.125
Exploit Attempt	-0.25
Cancel Exploit	-4
Change IP cost	-8

- **Attacker rewards and cost:** Table 3 shows the detailed reward structure for the attacker. The attacker gains a significant reward for successfully exploiting a vulnerability

Table 4: Defender Reward Design

Action	Reward
Successful Defense	100
Rate limit IP cost	-8
Rate limit Port Cost	-12
Close port cost	-40
Trap Set	-4
Block normal request	-8

and faces a penalty when falling into traps or failing exploits. Probing actions incur a small cost but can provide valuable information about vulnerabilities or honeypots.

- **Defender rewards and cost:** The defender’s rewards are linked to its defensive actions, such as setting traps or rate-limiting the attacker. Table 4 summarizes the defender’s reward design. The defender earns rewards for successfully blocking attacks, such as by trapping the attacker or rate-limiting its actions. However, penalties are imposed for defensive mistakes like blocking legitimate traffic or unnecessarily closing ports. [13]

4 Agent Training and Evaluation

We trained the agents in 2 environments enabling simple to complex strategies in the end. Each environment had the same technical scope but trained on different algorithms, parameters and scope for coming up with complex strategies.

4.1 Model Structure

In our model, we employ a Deep Q-Network for both the attacker and defender agents. This approach allows both agents to adapt their strategies based on the rewards received from their actions. [14] Both the attacker and defender use a sparse Q-table implementation, represented as a dictionary. This structure significantly reduces memory usage, which is crucial for handling large state spaces efficiently. Both agents update their Q-values after each action using the Q-learning update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \dots \dots \dots (5)$$

Here $Q(s_t, a_t)$ is the Q-value for the state-action pair (s_t, a_t) , r_t is the reward at time step t , and γ is the discount factor. The DQN maps the agent’s observation vector to Q-values for all possible discrete actions:

$$Q : \mathbb{R}^{d_{\text{obs}}} \rightarrow \mathbb{R}^{d_{\text{action}}}, \quad Q(s) = [Q(s, a_1), \dots, Q(s, a_{d_{\text{action}}})], \dots \dots \dots (6)$$

Here,

- d_{obs} is the dimension of the observation vector
- d_{action} is the number of discrete actions
- $s \in \mathbb{R}^{d_{\text{obs}}}$ is the current observation
- $Q(s, a)$ estimates the expected cumulative reward for taking action a in state s

Algorithm 1 Deep Q-Network (DQN) with ε -Greedy Policy

```

1: Initialize replay memory  $\mathcal{D}$ 
2: Initialize Q-network with random weights  $\theta$ 
3: Initialize target network  $\theta^- \leftarrow \theta$ 
4: for each episode do
5:   Initialize state  $s_0$ 
6:   for each step  $t$  do
7:     Choose action  $a_t$  using  $\varepsilon$ -greedy from  $Q(s_t; \theta)$ 
8:     Execute  $a_t$ , observe reward  $r_{t+1}$  and next state  $s_{t+1}$ 
9:     Store  $(s_t, a_t, r_{t+1}, s_{t+1})$  in  $\mathcal{D}$ 
10:    if enough samples in  $\mathcal{D}$  then
11:      Sample random minibatch from  $\mathcal{D}$ 
12:      Compute target:  $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$ 
13:      Update  $\theta$  to minimize  $(y - Q(s, a; \theta))^2$ 
14:    end if
15:    Periodically update target network:  $\theta^- \leftarrow \theta$ 
16:     $s_t \leftarrow s_{t+1}$ 
17:  end for
18: end for

```

Here the algorithm mainly represents how the model optimizes for best action after each timesteps with a balance of exploration and exploitation.

4.2 Parameters Used

Table 5: Primary hyperparameter (representative).

Hyperparameter	Value
Episodes	20000
Batch size	512
Replay buffer	75,000
T_{\min}	300 req
Normal requests per action	50-70
Ports Used N	10-15
Vulnerable ports	3-7
Trap detection	60%
Max previous history states	150

The large batch size of 512 samples leverages GPU parallelization for efficient neural network updates while providing stable gradient estimates that reduce learning variance. The substantial replay buffer capacity of 75,000 transitions maintains



Figure 3: Early training dynamics with 50-episode moving average rewards. The defender achieves rapid positive escalation while attacker rewards remain deeply negative, illustrating strong divergence in the initial learning phase.

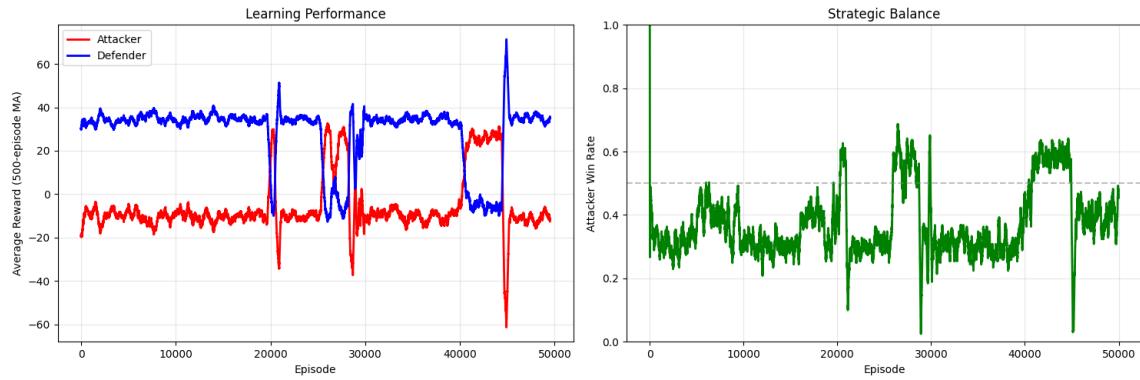


Figure 4: Learning Performance of Attacker and Defender Agents during initial training

diverse experience samples across different phases of strategy evolution, preventing catastrophic forgetting of previously effective tactics when agents adapt to new opponent behaviors. [15] The 150-state history window constrains the temporal complexity of strategic reasoning while providing sufficient context for multi-step attack sequences. This parameter directly impacts the computational complexity of state representation and influences the sophistication of temporal strategies both agents can develop. [16]

4.3 Evaluation

Figures 3, 4, and 5 together illustrate the learning dynamics and strategic balance between the defender and attacker over extended training. While the initial ones span long horizons, the latter emphasize adaptation to richer strategic repertoires.

Learning Dynamics: In the short-horizon view of Figure 3, the defender (blue) quickly achieves large positive returns (peaks near +60,000) within the first 200 episodes, while the attacker (red) falls sharply to negative values (40,000 to 60,000). This shows that even basic defenses like traps and rate-limiting yield immediate payoff, well before long-term stabilization.

Throughout extended training (Figure 4), defensive performance remains consistently positive ($\approx +30$ to $+40$ on average), with only brief transients, while the attacker fails to sustain gains. Strategic balance rarely exceeds parity, confirming defender dominance in 50,000 episodes. When agents are exposed to complex strategies (Figure 5), defender returns escalate dramatically (+300 to +600), and the attacker win rates stabilize well below parity. Richer defensive options, e.g. adaptive IP blocking, port-specific controls, and traps decisively tilt the equilibrium.

Together, these outcomes reveal a two-phase dynamic. Early training is characterized by steep and volatile divergences (Figure 3), with defenders establishing advantage almost immediately. Longer training smooths this volatility (Figure 4), while complex strategies (Figure 5) drive overwhelming defensive stability. [17] These findings highlight the role of reward shaping, training schedules, and defensive hierarchy in achieving stable convergence in adversarial reinforcement learning.

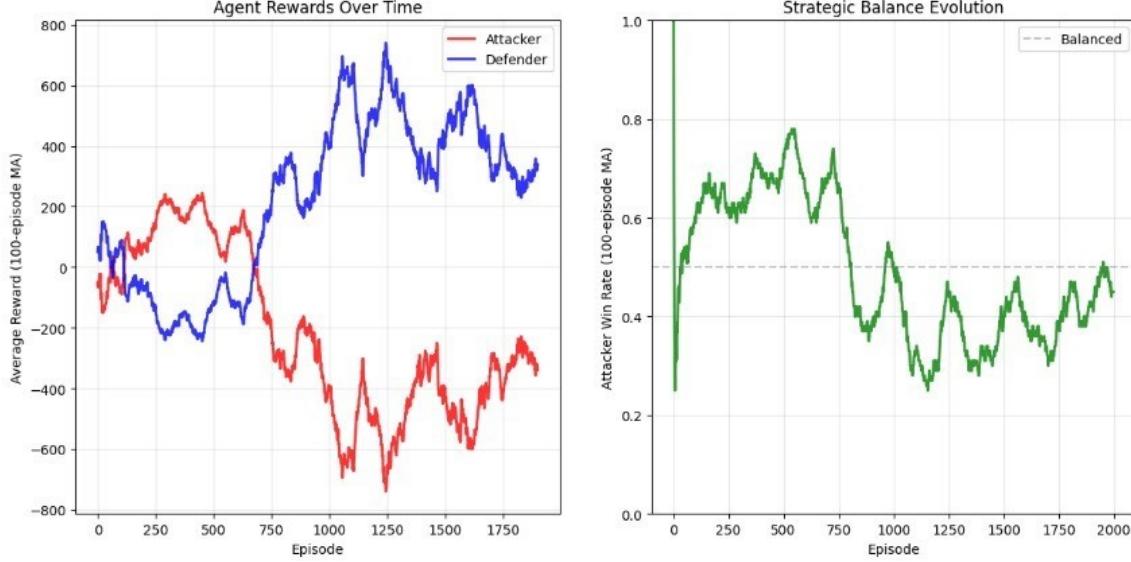


Figure 5: Learning Performance of Attacker and Defender Agents after adapting them to complex strategies

5 Results and Discussion

We summarize the principal findings, aggregated across random seeds and reported as mean values; variances are provided in supplementary logs. Results are organized around ablations, parameter sensitivities, and broader system-level insights.

5.1 Implementation Notes and Ablations

We evaluate both *raw* and *shaped* reward variants. Shaping accelerates convergence and reduces variance, but can also alter policy behavior. The false-positive penalty, modeled as a per-request cost when benign traffic is rate-limited, introduces a measurable availability–security tradeoff. Under the baseline configuration ($P_{trap} = 60\%$, $T_{min} = 300$), defenders converged to conservative strategies emphasizing port-level rate-limiting and selective traps. These policies achieved low false-positive rates while keeping attacker win rates near zero. Reward curves showed oscillations during early training that gradually stabilized with longer horizons. [18]

5.2 Effect of Trap Probability and Exploitation Thresholds

Trap probability has a strong effect on attacker viability. With $P_{trap} = 0\%$, attackers succeeded more frequently but defenders could still suppress them with aggressive rate-limiting, albeit at higher collateral costs. At the default $P_{trap} = 60\%$, attacker success declined substantially while defender costs remained moderate. Raising exploitation thresholds ($T_p = 400$) further reduced attacker success, though the increased sparsity

of terminal rewards slowed attacker learning. Reward shaping mitigated this effect by providing incremental progress signals, but ablations confirm that shaping must be tuned carefully to avoid policy bias.

5.3 Practical Insights

Three broader lessons emerge. First, defender observability provides a structural advantage: access to aggregated request histories and IP activity patterns enables robust discrimination of malicious from benign traffic. Second, reward design critically affects attacker learning under sparse feedback; shaping improves stability but risks biasing strategies. Third, training logistics matter: simultaneous learning introduces non-stationarity, while alternating updates, opponent populations, or centralized critics reduce instability. From a systems perspective, traps act as effective honeypot analogs when their detection probability is non-trivial, while selective rate-limiting remains a reliable mechanism for balancing security with service availability.

6 Conclusion

We designed and evaluated a compact adversarial reinforcement learning environment that captures key tradeoffs in brute-force exploits and reactive defenses. Our experiments demonstrate how trap mechanisms, exploitation difficulty, reward shaping, and training regimen influence outcomes. The findings emphasize the role of observability and reward engineering in multi-agent learning for security. Future work should explore centralized critics, opponent populations, richer attacker models, and transfer to more realistic network simulators.

References

- [1] World Economic Forum. Global Cybersecurity Outlook 2025. *Global Cybersecurity Outlook 2025* (Jan 2025). Accessed: Sep. 16, 2025. [Online]. Available: https://reports.weforum.org/docs/WEF_Global_Cybersecurity_Outlook_2025.pdf
- [2] Luanne Burns Chamberlain, Lauren Eisenberg Davis, Brian R. Gattoni and Martin Stanley. 2020. Automated Decision Systems for Cybersecurity and Infrastructure Security. *2020 IEEE Security and Privacy Workshops (SPW)*. Accessed: Sep. 16, 2025. [Online]. Available: <https://ieeexplore.ieee.org/document/9283870>
- [3] Mark Towers et al. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. Accessed: Sep. 16, 2025. [Online]. Available: <https://arxiv.org/pdf/2407.17032>
- [4] Greg Brockman et al. 2016. OpenAI Gym. Accessed: Sep. 16, 2025. [Online]. Available: <https://arxiv.org/pdf/1606.01540>
- [5] Majid Ghasemi et al. 2024. Introduction to Reinforcement Learning. Accessed: Sep. 16, 2025. [Online]. Available: <https://arxiv.org/abs/2408.07712>
- [6] Jianqing Fan et al. 2024. A Theoretical Analysis of Deep Q-Learning. Accessed: Sep. 16, 2025. [Online]. Available: <https://arxiv.org/abs/1901.00137>
- [7] Cassady, Caleb. 2022. Reinforcement Learning with Deep Q-Networks. *Masters Theses and Specialist Projects Paper 3554*. Accessed: Sep. 16, 2025. [Online]. Available: <https://digitalcommons.wku.edu/cgi/viewcontent.cgi?article=4558&context=theses>
- [8] Martijn Van Otterlo. 2012. Reinforcement Learning and Markov Decision Processes. *ResearchGate*, 2012. Accessed: Sep. 16, 2025. [Online]. Available: https://www.researchgate.net/publication/235004620_Reinforcement_Learning_and_Markov_Decision_Processes
- [9] Hariharan N and Paavai Anand G. 2022. A Brief Study of Deep Reinforcement Learning with Epsilon-Greedy Exploration. *International Journal of Computing and Digital Systems 11(1)*. Accessed: Sep. 16, 2025. [Online]. Available: https://www.researchgate.net/publication/357971826_A_Brief_Study_of_Deep_Reinforcement_Learning_with_Epsilon-Greedy_Exploration
- [10] Theresa Eimer et al. 2023. Hyperparameters in Reinforcement Learning and How To Tune Them. Accessed: Sep. 16, 2025. [Online]. Available: <https://arxiv.org/abs/2306.01324>
- [11] Patrick Phillips. 2021. Reinforcement Learning In Two Player Zero Sum Simultaneous Action Games. Accessed: Sep. 16, 2025. [Online]. Available: <https://arxiv.org/abs/2110.04835>
- [12] Rati Devidze. 2025. Reward Design for Reinforcement Learning Agents. Accessed: Sep. 16, 2025. [Online]. Available: <https://arxiv.org/abs/2503.21949>
- [13] Marek Grześ. 2017. Reward Shaping in Episodic Reinforcement Learning. *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems(2017)*, 565-573. Accessed: Sep. 16, 2025. [Online]. Available: <https://dl.acm.org/doi/10.5555/3091125.3091208>
- [14] Hooman Alavizadeh et al. 2022. Deep Q-Learning Based Reinforcement Learning Approach for Network Intrusion Detection. Accessed: Sep. 16, 2025. [Online]. Available: <https://www.mdpi.com/2073-431X/11/3/41>
- [15] Ildar Agliukov et al. 2022. A Method for Catastrophic Forgetting Prevention during Multitasking Reinforcement Learning. Accessed: Sep. 16, 2025. [Online]. Available: https://www.researchgate.net/publication/362688380_A_Method_for_Catastrophic_Forgetting_Prevention_during_Multitasking_Reinforcement_Learning
- [16] Lewis Hammond et al. 2021. Multi-Agent Reinforcement Learning with Temporal Logic Specifications. Accessed: Sep. 16, 2025. [Online]. Available: <https://arxiv.org/abs/2102.00582>
- [17] Mojtaba Rostami et al. 2021. Optimal Reinforcement Learning with Asymmetric Updating in Volatile Environments: a Simulation Study. Accessed: Sep. 16, 2025. [Online]. Available: https://www.researchgate.net/publication/349359103_Optimal_Reinforcement_Learning_with_Asymmetric_Updating_in_Volatile_Environments_a_Simulation_Study
- [18] Ahmad Muhammin Ismail et al. 2025. Toward Reduction in False Positives Just-In-Time Software Defect Prediction Using Deep Reinforcement Learning. Accessed: Sep. 16, 2025. [Online]. Available: <https://ieeexplore.ieee.org/document/10485281>