Advances in Knowledge-Based Systems, Data Science, and Cybersecurity Received 22-10-2025; Accepted 5-12-2025; Published 12-12-2025

Reinforcement Learning of Defensive Strategies Against Attacks in Partially Observable Local Area Networks

Takudzwa V. Banda takudzwa@sun.ac.za
Department of Mathematical Sciences
University of Stellenbosch
Stellenbosch, Cape Town, South Africa

Gavin B. Rens gavinrens@sun.ac.za
Department of Mathematical Sciences
University of Stellenbosch
Stellenbosch, Cape Town, South Africa

Corresponding Author: Takudzwa V. Banda

Abstract

Enterprise Local Area Networks (LANs) face increasing threats from self-propagating worm ransomware that disrupts operations and encrypts data. Traditional intrusion detection systems rely on static rules that fail against adaptive malware. Reinforcement Learning (RL) offers a promising approach for developing autonomous cybersecurity agents capable of learning optimal defensive strategies through interaction with their environment. However, existing studies often rely on simplified network models, single unreliable detectors, and overlook unsafe defensive actions that can cause data loss and network downtime. This study develops an adaptive RL-based defense framework in Microsoft's *CyberBattleSim*, formalized as a Partially Observable Markov Decision Process (POMDP). The framework integrates three key mechanisms: (1) a multi-detection system combining network- and node-level detectors, (2) probabilistic shielding to reduce unsafe actions and preserve data and network availability, and (3) a trust-based belief update mechanism that weights detector outputs. Experimental results demonstrate that multi detection system improves both training stability and defensive performance compared to single detection system. Probabilistic shielding minimizes data and network availability losses, both of which are key indicators of LAN resilience, while enabling the agent to learn an optimal policy. The trust system further enhances tactical precision by ensuring that belief updates accurately reflect detector reliability and confidence.

Keywords: Reinforcement Learning, Adversary Agent, Worm Ransomware, Cybersecurity, Partially Observable Markov Decision Process (POMDP), Shielding, Trust

1. INTRODUCTION

Enterprise Local Area Networks (LANs) are collections of interconnected devices within a small, confined geographic area, such as a home, office, or building [1]. LANs allow for resource sharing

Citation: Takudzwa V. Banda et al. Reinforcement Learning of Defensive Strategies Against Attacks in Partially Observable Local Area Networks. **Advances in Knowledge-Based Systems, Data Science, and Cybersecurity**.Research 2025;2(3):17.

Page No. 337-374

like printers, files, and internet access [1]. LANs are vulnerable to cyberattacks, particularly worm ransomware. This malware autonomously spreads, encrypts files, alters data, and disrupts normal operations [2–4]. Reports highlight its severity: ENISA 2024 lists it among top EU cybersecurity threats, Verizon DBIR 2025 recorded over 12,000 breaches with most involving ransomware, IBM estimates average breach costs at $4.44 million, and NETSCOUT reports 8.91 million global attacks in late 2024 [5–8].

Traditional Intrusion Detection and Prevention Systems (IDS/IPS) rely on static rules targeting specific threats and are insufficient against adaptive malware [9–12]. Reinforcement Learning (RL) enables agents to learn optimal defensive strategies through continuous interaction [13–18]. RL defenders can traverse LANs automatically to repair, patch, and restore infected nodes. Previous work trained RL agents in POMDP-simulated LANs [19–22], but real-world applicability is limited.

Prior RL approaches to cyber defense present several limitations. Many rely on oversimplified environments that ignore network interdependencies, node attributes, and realistic traffic patterns. This can produce RL agent decision making that do not generalize against sophisticated ransomware [23, 24]. Many approaches rely on a single detector, giving incomplete network visibility [25]. Critical LANs aspects such as data loss and network availability are often ignored, including consequences of hard resets or unnecessary node isolation [22]. They also reply on the assumption of perfect detector reliability that may mislead the agent when detectors provide incorrect observations [26].

To address these limitations, we use CyberBattleSim [19], with realistic network topologies, node attributes, and traffic patterns. The problem is formalized as a Partially Observable Markov Decision Process (POMDP), where the defender cannot directly observe the underlying state and detectors provide noisy observations. The defender must prevent attacks and remediate infected nodes in real time, while the attacker aims to infect as many nodes as possible and reach critical servers. Our approach improves RL cybersecurity agent training and enables optimal policy learning through three integrated mechanisms. First, a novel multi-detection system combines network-level and node-level detectors to provide a more complete view of the network, addressing limitations of single detectors. Second, a novel probabilistic shielding constrains unsafe actions, such as hard resets and unnecessary node isolation, preventing data loss and maintaining network availability. Third, a novel trust system for detectors in LAN evaluates the outputs of network and node-level detectors based on their historical reliability, and incorporates trust-based belief updates to maintain accurate probabilistic beliefs over hidden network states. However, other studies such as [27, 28] focuses on trust between agents, our approach applies trust directly to the detectors, allowing the RL agent to make more informed and reliable decisions even when observations are noisy or partially incorrect.

This study makes the following contributions. First, a multi-detection system that provides the RL agent with a more improved view of the network under partial observability. Second, probabilistic shielding that constrains unsafe actions to reduce potential data loss and network downtime that may occur during agent exploration. Third, a detectors' trust-based belief update system in LAN that evaluates their reliability over time and maintains accurate probabilistic beliefs over hidden states to enhance the agent's decision-making precision under uncertainty.

The remainder of this article is organized as follows. Section 2 reviews prior work and identifies key research gaps. Section 3 states research questions. Section 4 introduces foundational concepts including POMDPs, RL algorithms, trust mechanisms, and probabilistic shielding. Section 5

presents the proposed defender agent architecture, detailing the POMDP formulation, neural particle filtering for belief maintenance, multi detection system, trust system, and probabilistic shielding for safe exploration. Section 6 describes the experimental methodology, training procedures, validation protocols, evaluation metrics, and presents results. Section 7 interprets the experimental findings, providing mechanistic explanations for observed performance patterns and algorithm-specific behaviors. Section 8 summarizes the key contributions and outlines directions for future research.

2. RELATED WORK

LAN as POMDP. Cyber defense under uncertainty can be modeled as a Partially Observable Markov Decision Process (POMDP), where the network state is partially observed through detectors and actions affect system dynamics [19, 20, 29]. Authors in [29], and [30], showed that deep RL policies can learn effective defense strategies in these environments. Authors in [11, 31] demonstrated that learned environment models help policies adapt to dynamic attacks. However, many studies rely on hand-crafted transition and observation models with fixed probabilities and simple node dependencies. This limits realism because they cannot capture complex interactions between nodes, adaptive adversaries, or multi-detector uncertainty. Authors in [32] extended POMDPs to multi-agent RL for cooperative defense, but often treated nodes independently and separated safety from learning, reducing the effectiveness of coordinated defense. These limitations can produce overconfident policies and unsafe exploration. Our approach addresses these gaps by combining learned environment dynamics with trust-weighted probabilistic belief states, enabling robust multi-agent decision-making under partial observability.

Single detection systems. Authors in [18], and [25], relied on a single detector to observe network states, often aggregating all signals into a single estimate. This approach fails to capture the diversity of network signals and makes policies vulnerable to noise, missed detections, or compromised detectors. Authors in [33], showed that multi-detector fusion improves robustness in robotics, but integration into RL-based cyber defense is limited. The lack of multi-detector support in prior RL approaches reduces resilience and increases the risk of unsafe actions when detector readings are unreliable. Our work maintains trust-weighted beliefs over multiple detectors, allowing the agent to discount unreliable signals and make robust decisions under partial observability.

Trust in detectors. Authors in [28], and [27], studied trust in multi-agent systems, usually between agents. Authors in [34], explored decentralized trust frameworks, but these frameworks are applied only to agent-to-agent interactions, not to detectors in LAN cyber defense. Authors in [18, 25], assume all detectors are perfectly reliable, giving full trust to every reading. This assumption can lead to poor policy performance when detectors are noisy. In contrast, we apply trust directly to each detector, enabling the agent to weight signals adaptively and ignore unreliable or adversarial observations during belief updates and action selection. This explicitly addresses a major gap in current RL-based cyber defense research.

Probabilistic shielding and safe RL. Current RL-based cyber defense research often ignores LAN operational principles, allowing agents to "win the game" by taking actions that cause unnecessary data loss or network downtime, such as hard resets or isolating nodes without cause.

When studies do consider operational constraints, they rely on hard-coded safety rules, such as in [32], which prevent unsafe actions but limit exploration and ignore uncertainty in state estimates. Such approaches fail to balance learning effectiveness with operational safety. Probabilistic shielding [35], provides a flexible mechanism, guiding the agent toward actions that are both effective and safe. Our work integrates shielding with trust-weighted beliefs over multiple detectors, allowing agents to respect LAN principles, reduce data loss, maintain network availability, and still learn efficiently under partial observability.

3. RESEARCH QUESTIONS

To guide the research gaps investigation, we pose the following research questions:

RQ1: *How does a multi-detector system impact an RL agent's learning performance compared to a single-detector system in partially observable network defense scenarios?*

RQ2: *How well does probabilistic shielding help RL agents explore safely and prevent data or availability loss without hurting their performance?*

RQ3: *How do trust mechanisms for detectors improve agent learning under noisy detector observations, and how does trust interact with shielding to enhance policy safety and effectiveness?*

4. PRELIMINARIES

This section introduces the fundamental concepts underlying our approach. The details of specific implementations are provided in Section 5.

4.1 Partially Observable Markov Decision Process (POMDP)

A POMDP extends an MDP by introducing uncertainty in state observability, defined as $\langle S, A, O, T, \Omega, R, \gamma \rangle$, where S is the state space, A is the action space, O is the observation space, $T(s, a, s')$ is the transition function, $\Omega(s', a, o)$ is the observation function, $R(s, a)$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor [36]. Since the true state s is not directly observable, the agent maintains a belief state b, which is the probability distribution over possible states, updated via Bayes' rule:

$$b'(s') = \eta \cdot \Omega(s', a, o) \sum_{s \in S} T(s, a, s') b(s), \tag{1}$$

where η is a normalization constant. The policy $\pi(b)$ maps beliefs to actions to maximize expected cumulative discounted return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$.

4.2 Single and Multi-Detection Systems

In cybersecurity and network monitoring, agents often rely on detectors to observe the environment. A *single-detector* system uses one source of information to estimate network state. In contrast, a *multi-detection* system aggregates observations from several detectors, improving accuracy and robustness. Formally, let o_i denote the observation from detector i. In a multi-detection system, the belief state update can be expressed as:

$$b'(s') = \eta \sum_i P(o_i \mid s') \sum_{s \in S} T(s, a, s') b(s), \tag{2}$$

where η is a normalization constant, $T(s, a, s')$ is the transition function, and $b(s)$ is the prior belief over states. This aggregation allows the agent to reduce the effect of noisy or unreliable detectors. In our work, we used a multi-detector setup where BeliefNet combines the outputs of multiple detectors into a probabilistic estimate of node states. This improves belief estimation compared to relying on a single detector [25], enabling more robust decision-making under partial observability.

4.3 Reinforcement Learning Algorithms

Standard RL algorithms (DQN [37], A2C [37], PPO [38]) are designed for fully observable MDPs where agents have direct access to state s. In POMDPs, these algorithms operate on belief representations rather than true states. We used three algorithms:

- **Deep Q-Learning (DQN):** Learns action-value function $Q(b, a)$ using neural networks with experience replay and target networks to stabilize training [37].
- **Advantage Actor-Critic (A2C):** Combines policy gradients (actor) with value estimation (critic) using advantage function $A(b, a) = Q(b, a) - V(b)$ to reduce variance [37].
- **Proximal Policy Optimization (PPO):** Constrains policy updates via clipped objective $L^{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)]$ to ensure stable learning [38].

4.4 Trust and Shielding Mechanisms

Trust. A trust score $T \in [0, 1]$ quantifies the reliability of an information source based on historical accuracy [39, 40]. In our work, trust is used to weight the contributions of different detectors when updating the belief state over network nodes. This allows the agent to rely more on accurate detectors and mitigate the influence of noisy detectors. In contrast, prior studies in multi-agent RL primarily use trust to weight agent interactions or to guide cooperation [34, 35], whereas we adapt it for detectors and belief update.

Shielding. A shield defines a set of safe actions $A_{\text{safe}}(s) \subseteq A(s)$ for each state s, preventing unsafe behaviors during learning [41]. We implement probabilistic shielding by assigning safety probabilities $P(\text{safe}|s, a)$ to candidate actions, which guides the RL agent towards safer interventions while

still allowing exploration. Compared to prior work, which typically uses hard coded rules to strictly prevent unsafe actions during exploration or execution [35, 42, 43], our approach uses probability shielding to soft reduce selection of unsafe actions.

4.5 Learned Transition and Observation Models

Classical POMDP methods such as value iteration and policy iteration) assume known T and Ω models [36]. In adversarial cyber defense, these models are unknown and analytically intractable due to high-dimensional state spaces, complex node interactions, and adaptive adversaries. Prior RL approaches either use handcrafted models [29] or ignore belief maintenance entirely [12].

BeliefNet ($\mathcal{B}_{\text{net}}$): Our observation model. Instead of specifying $\Omega(s', a, o)$ analytically, BeliefNet learns the likelihood $P(o|s)$ using a neural network trained on detector confidence distributions. Given state s and observation o, BeliefNet outputs $\hat{p} = \mathcal{B}_{\text{net}}(s, o)$, approximating how likely observation o would be under state s. This is trained via cross-entropy loss against probabilistic labels derived from detector softmax outputs:

$$\mathcal{L}_{\text{belief}} = -\sum_{j} p^{(j)} \log \mathcal{B}_{\text{net}}(s, o^{(j)}), \tag{3}$$

where $p^{(j)}$ is the target probability from detector confidences.

StateNet ($\mathcal{S}_{\text{net}}$): Our transition model. Instead of handcrafting $T(s, a, s')$, StateNet learns next-state prediction: $\hat{s}_{t+1} = \mathcal{S}_{\text{net}}(s_t, a_t)$. Trained via mean squared error:

$$\mathcal{L}_{\text{state}} = \mathbb{E}[\|\mathcal{S}_{\text{net}}(s_t, a_t) - s_{t+1}\|^2], \tag{4}$$

using true state transitions observed during simulation.

Integration with Particle Filtering: BeliefNet and StateNet replace the prediction and weighting steps in standard particle filtering [44]:

1. *Prediction:* $s_t^{(i)} = \mathcal{S}_{\text{net}}(s_{t-1}^{(i)}, a_{t-1}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$ adds process noise.
2. *Weighting:* $w_t^{(i)} = \mathcal{B}_{\text{net}}(s_t^{(i)}, o_t)$, then normalize $w_t^{(i)} \leftarrow w_t^{(i)} / \sum_j w_t^{(j)}$.
3. *Resampling:* Draw N particles proportionally to weights.

This approach enables model-free belief maintenance: no handcrafted transition or observation functions required. BeliefNet and StateNet learn environment dynamics directly from experience, making our architecture scalable to envrionment where explicit POMDP specifications are infeasible.

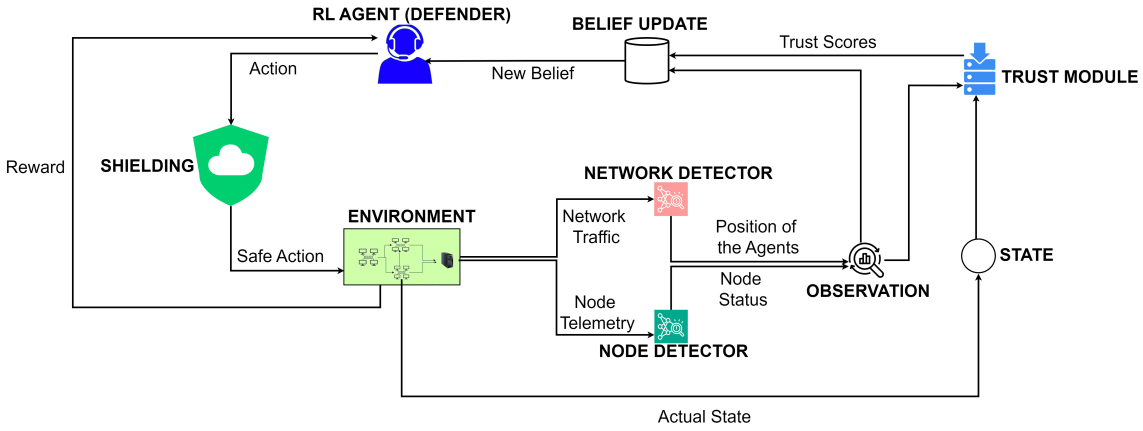

Figure 1: RL defender agent architecture

5. PROPOSED REINFORCEMENT LEARNING DEFENDER ARCHITECTURE

Our RL agent operates as a belief-based reinforcement learning system with integrated shielding mechanisms, trust system, and multi detection system (see FIGURE 1). At the core, the *RL agent (Defender)* (hereafter referred to as the *Blue agent* and adversary as the Red agent) receives belief states rather than direct environment observations and selects defensive actions based on learned policies. To integrate the safety mechanism, we implemented a *Shielding module* that acts as a safety layer between the agent and environment, evaluating proposed actions for safety before execution and down-weighting unsafe actions to ensure high-probability of safe actions reach the environment. We further designed a *Multi detection system* that comprises two parallel detectors: a *Network detector* monitoring network traffic and agent positions, and a *Node detector* observing node telemetry and status information. These detectors produce observations that feed into the belief update system. We added a *Trust module* that maintains and updates trust scores for each detector based on their historical accuracy, weighting detector observations according to their reliability. Finally, the *Belief update system* then maintains a probabilistic belief over the actual environment state by fusing multi-detector observations weighted by trust scores. Rather than relying on handcrafted models, it uses learned neural network approximations of the transition function and observation function within a particle filtering framework to update beliefs and provide the Blue agent with belief states rather than raw observations. Our architecture operates as a closed loop: environment states generate detector observations, which are weighted by trust scores and fused into belief representations; the Blue agent processes these beliefs to propose actions; the shielding module filters these actions for safety; safe actions execute in the environment; and reward signals (potentially modified by shielding) return to the agent for policy learning.

5.1 Multi Detection System

In the proposed architecture, the Blue agent perceives the environment through machine learning–based detectors. These detectors formalise the observation process in the POMDP formulation, where the true system state is hidden and must be inferred probabilistically [45, 46]. To overcome the limitation of single-detector reliance, we integrated both network and node detectors. Two detector provide the Blue agent with probabilistic observations. In order to understand how the detectors work and what information they will process, we first need to define in detail node attributes and network traffic, two crucial elements for these detectors.

Node Attributes. Each node was instrumented with attributes reflecting its configuration and operational state. These included:

- Exposed services: Secure Shell (SSH), Remote Desktop Protocol (RDP), Hypertext Transfer Protocol (HTTP/HTTPS), File Transfer Protocol (FTP), Server Message Block (SMB), My Structured Query Language (MySQL), and Internet Control Message Protocol ping (PING) [46, 47].
- Exploitable vulnerabilities: enabling local and remote attacks such as privilege escalation, credential leakage, denial of service, lateral movement, file encryption, and SQL injection [48].
- Node-level indicators: CPU usage, memory utilisation, disk space, registry keys, and file system integrity [49].
- Red agent flag: a binary indicator of whether the node is infected.

The core observable attribute vector $\mathcal{A}_n$ for each node's status estimation is defined as

$$\mathcal{A}_n = \{u_n, m_n, d_n, f_n, r_n, fw_n, v_n, sf_n, rf_n, x_n\}, \tag{5}$$

where u_n denotes CPU usage, m_n the memory utilisation, and d_n the available disk space. The variable f_n captures the file system state (either `access` or `encrypted`), while r_n is a binary indicator of Red agent presence. Structural security properties are represented through fw_n for firewall configuration, v_n for exploitable vulnerabilities, sf_n for system files, and rf_n for registry entries. Finally, x_n expresses the machine's overall operational state, which may be `Running` or `Stopped`.

Network Traffic. In addition, the environment was extended with a traffic generator [46] to emulate legitimate background communications between nodes. Each traffic entry, T_x, contained features, x, such as the source node, which specifies where the traffic originates; the destination node, which indicates where the traffic is sent; the service type, describing the protocol or service used to transmit the traffic; and the encryption status, which identifies whether the traffic is encrypted or plaintext [46]. Now that we have explained node attributes and network traffic, we can explain our detectors.

1. **Node Detector:** The Node Detector $\mathcal{D}_{\text{node}}$ is a probabilistic logic function that estimates the operational status of each node n by analysing the combination of values in its attribute vector

$\mathcal{A}_n$, defined in Equation 5, and mapping them to an estimated status $\hat{D}_{\text{node}}$ from the discrete set S_{node}. This is formalised as follows:

$$\mathcal{D}_{\text{node}} : \mathcal{A}_n \longmapsto \hat{D}_{\text{node}}, \quad \hat{D}_{\text{node}} \in S_{\text{node}},$$

where

$$S_{\text{node}} = \{\text{Normal, UnsecuredFirewall, UnsecuredVul, At risk,}$$
$$\text{Compromised, Critical, Encryption, Stopped}\}.$$

Here, `Normal` denotes a healthy node, `UnsecuredFirewall` or `UnsecuredVul` indicate exposure through weak firewall rules or exploitable vulnerabilities, `At risk` captures combined weak firewall rules and exploitable vulnerabilities, `Compromised` reflects confirmed malware or attacker presence, `Critical` represents severe degradation where malware is confirmed present with weak firewall rules and exploitable vulnerabilities, `Encryption` indicates ransomware-induced file encryption, and `Stopped` denotes a node that is completely inactive. To handle uncertainty and nonlinear attribute interactions, the detector $\mathcal{D}_{\text{node}}$ implements a probabilistic scoring mechanism. It first assigns rule-based scores to each candidate status $s_{\text{node}} \in S_{\text{node}}$ based on the node's attributes $\mathcal{A}_n$. These scores reflect the degree of evidence supporting each potential status.

These scores are then transformed into a probability distribution using a softmax function [50, 51]:

$$P(s_{\text{node}} \mid \mathcal{A}_n) = \frac{e^{\text{score}(s_{\text{node}} \mid \mathcal{A}_n)}}{\sum_{s'_{\text{node}} \in S_{\text{node}}} e^{\text{score}(s'_{\text{node}} \mid \mathcal{A}_n)}} \tag{6}$$

This posterior probability quantifies how likely node n is to be in status s_{node}, given the current values of its attributes $\mathcal{A}_n$. The detector also computes the prediction *confidence level*, which is defined as the highest probability assigned among all status classes. In other words, the confidence level is simply the maximum probability from this distribution [50, 51]:

$$\text{conf}_n = \max_{s_{\text{node}} \in S_{\text{node}}} P(s_{\text{node}} \mid \mathcal{A}_n) \tag{7}$$

It quantifies how strongly the detector supports the most likely status, given the available attribute evidence, and is subsequently used to weight observation likelihood during belief updates and when training the observation model. The final estimated node status $\hat{\mathcal{D}}_{\text{node}}$ is determined by selecting the class $s_{\text{node}} \in S_{\text{node}}$ that maximizes the posterior probability:

$$\hat{\mathcal{D}}_{\text{node}} = \arg \max_{s_{\text{node}} \in S_{\text{node}}} P(s_{\text{node}} \mid \mathcal{A}_n)$$

Example: Status Prediction. To illustrate the node-level inference process, consider a simplified telemetry snapshot from a Windows Workstation node:

```
cpu_usage = 91
memory_usage = 88
disk_space = 45
```

```
files_status = "encrypted"
red_agent_installed = True
machine_status = Running
firewall_config = "ALLOW:HTTP,FTP,RDP"
vulnerabilities = ["REMOTE", "LOCAL"]
system_files = ["malware.exe", ...]
registry_keys = {"HKLM": "compromised"}
```

Using Equation (6) for the scoring function, each candidate status is assigned a score based on this telemetry. After normalisation, the detector outputs the following probabilities:

$$\begin{aligned} P(\text{Normal}) &= 0.0001, \\ P(\text{UnsecuredFirewall}) &= 0.0003, \\ P(\text{UnsecuredVul}) &= 0.0004, \\ P(\text{At risk}) &= 0.0010, \\ P(\text{Compromised}) &= 0.0012, \\ P(\text{Critical}) &= 0.015, \\ P(\text{Encryption}) &= 0.982, \\ P(\text{Stopped}) &= 0.016. \end{aligned}$$

The most probable class is therefore:

$$\hat{\mathcal{D}}_{\text{node}} = \texttt{Encryption}, \quad \text{conf}_n = 0.982$$

This outcome reflects a high-confidence detection of an ongoing encryption event consistent with ransomware behaviour.

Network Detector. The Network Detector $\mathcal{D}_{\text{net}}$ generates two hidden state components from traffic logs: (i) the Red agent's current position and (ii) the node under attack. The $\mathcal{D}_{\text{net}}$ architecture comprises three XGBoost machine learning models [52], trained offline on synthetic traffic generated from CyberBattleSim simulations during the pre-training phase of RL agents. These logs emulate both normal and malicious traffic within a controlled LAN environment, ensuring consistency between the detector and RL policy training. The sample of the dataset can be found here `https://github.com/VinceTaku/RL_training_experiment_results/blob/main/network_traffic.csv`.

A comparative evaluation was performed across four classifiers—XGBoost, Support Vector Machine (SVM), Random Forest (RF), and a Feedforward Neural Network (FNN)—following [53]. XGBoost achieved the highest accuracy across all detection tasks, confirming its suitability for this system.

Each traffic entry T_x passes through this multi-stage pipeline: $\mathcal{D}_{\text{mal}}$ first flags malicious activity, after which $\mathcal{D}_{\text{red}}$ and $\mathcal{D}_{\text{att}}$ infer the adversary's location and target respectively.

Confidence Level Outputs. Given a traffic feature vector x, $\mathcal{D}_{\text{red}}$ and $\mathcal{D}_{\text{att}}$ produce unnormalised logits $f(x) = [z_1, \ldots, z_K]$, where K is the number of nodes. Probabilities are obtained using softmax:

Table 1: Network Detector sub-modules and their purpose.

Detector	Role
$\mathcal{D}_{\text{mal}}$ (Malicious Traffic)	Binary classifier; distinguishes benign from malicious traffic.
$\mathcal{D}_{\text{red}}$ (Red agent Position)	Multi-class; estimates attacker location in the LAN.
$\mathcal{D}_{\text{att}}$ (Node Under Attack)	Multi-class; predicts which node is being targeted.

$$\text{Conf}(n_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \quad i = 1, \ldots, K. \tag{8}$$

These distributions satisfy $\sum_{i=1}^{K} \text{Conf}(n_i) = 1$ and quantify the detector's confidence over possible Red agent positions and attack targets. The resulting confidence vectors are integrated into the observation model during belief updates.

5.2 Problem Formulation

The Blue agent decision process is formalized as a POMDP, introduced earlier, $\mathcal{M} = \langle S, A, O, T, \Omega, R, \gamma \rangle$, where the agent operate under uncertainty and receive noisy, incomplete observations of the environment. In addition, in this work adopted a *model-free* formulation. That is, T and Ω are not known a priori, but are instead approximated directly from data using neural networks, enabling the agents to learn effective policies without relying on handcrafted models of the environment. This ensures scalability to complex and realistic network scenarios where explicit models are analytically intractable. This implementation integrates deep learning techniques inspired by *Deep Particle Filter Networks (DPFN)* [54]

The Blue agent's action space is

$$A_{\text{blue}} = \left\{ \begin{array}{l} \texttt{patch\_node}, \\ \texttt{update\_firewall\_rules}, \\ \texttt{isolate\_node}, \\ \texttt{reset\_node}, \\ \texttt{restore\_node}, \\ \texttt{decrypt\_node}, \\ \texttt{lateral\_move}, \\ \texttt{secure\_harden\_node}, \\ \texttt{do\_nothing} \end{array} \right\}.$$

The Blue agent perceives the system through node and network detectors:

$$O_{\text{blue}} = \left\{ \begin{array}{l} \texttt{pos\_blue}: \text{ Blue agent position}, \\ \texttt{pos\_red}: \text{ Red agent position from } \mathcal{D}_{\text{red}}, \\ \texttt{node\_att}: \text{ Target node from } \mathcal{D}_{\text{att}}, \\ \texttt{n\_stat\_att}: \text{ Status of target node}, \\ \texttt{n\_stat\_pos\_red}: \text{ Red pos. status from } \mathcal{D}_{\text{node}}, \\ \texttt{n\_stat\_pos\_blue}: \text{ Blue pos. status from } \mathcal{D}_{\text{node}} \end{array} \right\}.$$

A typical single detector will have the Blue agent perceives the system as:

$$O_{\text{blue}} = \left\{ \begin{array}{l} \texttt{pos\_blue}: \text{ Blue agent position}, \\ \texttt{n\_stat\_pos\_red\_inferred}: \text{ Inferred Red pos. status from } \mathcal{D}_{\text{node}}, \\ \texttt{n\_stat\_pos\_blue}: \text{ Blue pos. status from } \mathcal{D}_{\text{node}} \end{array} \right\}.$$

The single detector produces an observation with node status and the blue agent position, but lacks network-level activities. Crucially, the defender does not receive direct adversary position (`pos_red`); instead, it must rely on simple status changes (`n_stat_pos_red_inferred`) to infer the Red agent's location, making it fragile and prone to delayed reactions [12, 30, 55, 56]. The transition model specifies the probability of moving between states given an action. In practice, the full transition dynamics are unknown and analytically intractable due to high-dimensional dependencies. Therefore, we approximate T using a neural network, enabling a model-free approach that learns transition patterns directly from experience. The observation model defines the probability of an agent receiving a particular observation given the true state. For the Blue agent, this is informed by the confidence distributions of $\mathcal{D}_{\text{node}}$ and $\mathcal{D}_{\text{net}}$. As with T, the observation process is complex and uncertain. We therefore parameterise Ω with a neural network, allowing the agent to learn an observation likelihood function from data rather than relying on handcrafted rules. This model-free treatment ensures scalability to more realistic environments where explicit models are unavailable. In a simplified way, the Blue agent is rewarded for restoring nodes to a Normal state, slowing or stopping the Red agent's progress, and maintaining the availability of critical services, and penalised for the exact opposite. A discount factor $\gamma \in (0, 1)$ ensures that the blue agent prioritise fast progression toward its goals by seeking to neutralise threats quickly.

5.3 Observation Model (BeliefNet)

The observation model Ω is not explicitly defined but learned through a neural network we refer to as *BeliefNet* ($\mathcal{B}_{\text{net}}$). Its purpose is to approximate the likelihood of receiving an observation o_t given the true hidden state s_t. BeliefNet is critical for particle filtering, as it assigns weights to particles based on how well they explain the observed detector outputs.

$$\mathcal{B}_{\text{net}} \approx P(o_t \mid s_t).$$

At each simulation step t, the simulator provides access to the true system state s_t. In parallel, detectors for the Blue agent produce confidence scores over candidate observation attributes using Equations 6, 7, and 8. These confidences define a probability distribution over possible observations that could arise from state s_t. Specifically, for each attribute (for example node status, Red position, attack target), the detector outputs a categorical distribution. By taking the Cartesian

product over all attributes, we construct a set of candidate observations $\{o_t^{(j)}\}_{j=1}^{M}$, each with an associated probability $p^{(j)}$ computed as the product of individual attribute probabilities (assuming conditional independence given s_t). These probabilities are normalised to sum to one. The result is a training triplet $(s_t, o_t^{(j)}, p^{(j)})$, which expresses how likely each candidate observation $o_t^{(j)}$ would be under the true state s_t, according to the detectors' confidence distributions. Importantly, $p^{(j)}$ serves as a probabilistic label reflecting observation likelihood, not a binary ground truth.

$\mathcal{B}_{\text{net}}$ is trained in an offline manner at the end of each simulation episode. During an episode, all triplets $(s_t, o_t^{(j)}, p^{(j)})$ are accumulated into a replay buffer. After the episode concludes, BeliefNet is updated using mini-batches sampled from this buffer, optimised with stochastic gradient descent (Adam optimizer, learning rate 10^{-4}). This episodic, replay-style training ensures that the networks generalise across diverse trajectories and do not overfit to the most recent observations, while still capturing the evolving attack and defence dynamics. BeliefNet is trained in a supervised manner to minimise the divergence between its predicted probabilities $\hat{p}^{(j)} = \mathcal{B}_{\text{net}}(s_t, o_t^{(j)})$ and the target probabilities $p^{(j)}$ obtained from detector confidences. By minimising this loss, BeliefNet learns to predict observation likelihoods that align with the detector's probabilistic assessments, effectively learning $P(o|s)$ from data. This is achieved using a cross-entropy loss:

$$\mathcal{L}_{\text{belief}} = -\sum_{j} p^{(j)} \log \hat{p}^{(j)}.$$

BeliefNet is implemented as a fully connected feedforward neural network with three hidden layers of dimensions [256, 128, 64], ReLU activations, and a final softmax output layer. The input is a concatenated vector representation of the state s_t and observation $o_t^{(j)}$, and the output is a scalar probability $\hat{p}^{(j)}$.

5.4 Learning State Transitions (StateNet)

In classical POMDPs, belief updates rely on a predefined transition model $T(s' \mid s, a)$, which specifies the probability of moving from a state s to a next state s' given an action a. In realistic, adversarial cybersecurity settings, such a model is unknown or infeasible to specify explicitly. To address this, we introduce *StateNet* $\mathcal{S}_{\text{net}}$, a neural function approximator that replaces the hand-crafted transition model. Blue agent maintains its own variant called *BlueStateNet* for defensive transitions. This network learn how system states evolve in response to the respective agent's actions. During simulation, we record triples (s_t, a_t, s_{t+1}), where s_t is the true current state obtained from the simulator, a_t the action taken by the Blue agent), and s_{t+1} the resulting true next state after the action's execution. $\mathcal{S}_{\text{net}}$ is trained to minimise prediction error between their estimated next state and the observed next state. The loss is defined as:

$$\mathcal{L}_{\text{state}}(\theta) = \mathbb{E}\left[\|\mathcal{S}_\theta(s_t, a_t) - s_{t+1}\|^2\right],$$

where $\mathcal{S}_\theta$ denotes StateNet parameterised by θ.

StateNet is implemented as a fully connected feedforward neural network with four hidden layers of dimensions of *512, 256, 128, and 64*, using ReLU activations. The input is a concatenated vector of

the current state s_t (encoded as a fixed-size vector) and a one-hot encoding of action a_t. The output is a predicted next state vector $\hat{s}_{t+1}$. Training is performed offline at the end of each episode using accumulated state-action-next-state samples stored in a replay buffer. Mini-batches of size 128 are sampled randomly, and the network is updated using the Adam optimizer with learning rate 10^{-4}. This episodic training allows the networks to generalise across diverse trajectories, capturing both attack dynamics and defensive interventions, without requiring explicit knowledge of true transition probabilities.

During particle filtering, StateNet is used to propagate particles forward: given a particle state $s_t^{(i)}$ and action a_t, the predicted next state is $s_{t+1}^{(i)} = \mathcal{S}_{\text{agent}}(s_t^{(i)}, a_t)$. To account for stochasticity in transitions, Gaussian noise with standard deviation $\sigma = 0.1$ is added to continuous state components after prediction. StateNet requires sufficient training data before it can reliably predict transitions. During the first 50 episodes, particles are propagated using the simulator's ground truth transitions with added noise to maintain uncertainty. After 50 episodes, StateNet predictions are used exclusively. This warm-up period ensures stable belief updates during early training when the transition model is poorly calibrated.

5.5 Particle Filtering and Belief Update

Particle filtering is used to maintain a probabilistic belief b_t over hidden states. We maintain $N = 500$ particles for computational efficiency. At each time step t, the particle filter performs the following operations:

1. *Prediction (for each particle $i = 1, \ldots, N$):*

$$s_t^{(i)} = \mathcal{S}_{\text{net}}\big(s_{t-1}^{(i)}, a_{t-1}\big) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2 I)$$

   where $\mathcal{S}_{\text{net}}$ is the learned StateNet and ϵ_i represents process noise.

2. *Weighting (for each particle $i = 1, \ldots, N$):*

$$w_t^{(i)} = \mathcal{B}_{\text{net}}\big(s_t^{(i)}, o_t\big)$$

   where $\mathcal{B}_{\text{net}}$ is the learned BeliefNet that outputs observation likelihood.

3. *Normalization:*

$$Z = \sum_{j=1}^{N} w_t^{(j)}, \quad \text{then for } i = 1, \ldots, N: \; w_t^{(i)} \leftarrow \frac{w_t^{(i)}}{Z}$$

4. *Resampling:* Draw N particles proportionally to $\{w_t^{(i)}\}_{i=1}^{N}$ using systematic resampling [44].

This procedure removes the need for hand-crafted transition or observation models. Instead, $\mathcal{S}_{\text{net}}$ and $\mathcal{B}_{\text{net}}$ approximate T and Ω from experience, enabling belief updates in high-dimensional and adversarial settings.

5.6 Probabilistic Shielding

A key challenge in reinforcement learning for cyber defence is that agents may take actions that, while technically valid, undermine the defender's primary objectives of preserving network availability and preventing data loss. Prior work often ignored this risk, leaving RL policies vulnerable to catastrophic behaviour. To address this gap, we implement *Probabilistic Shielding* as a soft safety layer that down-weights unsafe actions during policy selection [41] .

The shield assigns each action a probability of safety $P(\text{safe} \mid s, a)$ given the current state and action. This probability scales both the policy distribution and the reward:

$$\pi_{\text{shielded}}(a \mid s) \propto \pi_{\text{base}}(a \mid s) \cdot P(\text{safe} \mid s, a),$$

$$r_{\text{shielded}} = r \cdot P(\text{safe} \mid s, a).$$

Here, $\pi_{\text{base}}(a \mid s)$ is the original policy distribution output by the RL algorithm, $P(\text{safe} \mid s, a)$ is the safety probability computed by the shield, $\pi_{\text{shielded}}(a \mid s)$ is the adjusted safe policy (renormalised after scaling), r is the original reward, and r_{shielded} is the safety-aware reward.

The shield modifies the learning signal by directly altering received rewards to include a cost for unsafe actions, implicitly guiding the agent to avoid certain states or actions. This differs from traditional reward shaping, which provides intermediate rewards for desirable behaviours; here, we penalise undesirable behaviours proportionally to their risk [41]. Two actions were treated as unsafe due to their direct impact on system reliability:

- `isolate_node`: reduces network availability by disconnecting nodes.
- `reset_node`: lead to data loss by deleting or erasing system files.

Probabilistic Logic Shielding [41] formulates safety as a set of *probabilistic rules*. Each rule encodes a property such as *"resetting a node is unsafe"* and assigns a probability to capture the uncertainty of that violation. In the original formulation, probabilities are attached to logical clauses, often derived from domain knowledge or formal system models.

In this work, we adapt this principle to the cyber defence setting by grounding safety probabilities in the *node detector confidence levels* (Equation 7), which reflect the detector's certainty about the node's current status. Let $c_n \in [0, 1]$ denote the node detector's confidence that node n is in a risky status *Critical*, *Encryption*, or *Stopped*. FIGURE 2 is the safety probability, which is defined as a piecewise function that considers both the node status and action type, where $\beta_1 = 0.45$ and $\beta_2 = 0.5$ are hyperparameters controlling the strength of the safety constraint for isolate and reset node respectively. Higher values of β result in stronger discouragement of unsafe actions. The exponential decay ensures that:

- Actions on nodes flagged with high confidence ($c_n \to 1$) are strongly discouraged
- Actions on uncertain nodes ($c_n \approx 0$) remain available with minimal penalty

$$P(\text{safe} \mid s, a, n) = \begin{cases} e^{-\beta_1 \cdot c_n}, & \text{if } a = \texttt{isolate\_node}, \\ e^{-\beta_2 \cdot c_n} \cdot \mathbb{1}[\text{status}_n \notin \{\text{Critical, Encryption, Stopped}\}], & \text{if } a = \texttt{reset\_node}, \\ 1.0, & \text{otherwise.} \end{cases}$$

Figure 2: Safety probability definition used in the Probabilistic Shielding mechanism.

5.7 Trust Modelling

We incorporate *detector trust* into the belief update. The central idea is that detectors are imperfect, and their reliability should evolve over time based on observed accuracy rather than being assumed constant. Trust modelling provides a mechanism to weight detector contributions proportionally to their performance overtime [36, 39].Trust scores are maintained for three detector classes: node status ($\mathcal{D}_{\text{node}}$), Red agent position ($\mathcal{D}_{\text{red}}$), and attack target ($\mathcal{D}_{\text{att}}$). Trust scores are initialised at $T_0 = 0.1, T_0 = 0.15$, ,and $T_0 = 0.05$ for ($\mathcal{D}_{\text{node}}$), ($\mathcal{D}_{\text{red}}$), and ($\mathcal{D}_{\text{att}}$) respectively, representing initial uncertainty about detector reliability. At every time step t, each detector's prediction is compared against the ground truth state (available from the simulator for training purposes). Let $c_t \in \{0, 1\}$ denote whether the detector's prediction was correct:

$$c_t = \mathbb{1}[\text{detector prediction} = \text{true state attribute}]$$

Trust is updated using a logistic growth process:

$$T_{t+1} = T_t + \beta T_t \, (1 - T_t) \, c_t,$$

where $\beta = 0.05$ controls the growth rate. This ensures that:

- Trust grows slowly with each correct prediction
- Growth rate slows as trust approaches 1.0 (saturation effect)
- Incorrect predictions ($c_t = 0$) result in no update, preventing trust decay but slowing growth
- Trust remains bounded in $[0, 1]$

This conservative update rule reflects that trust should be earned gradually through consistent accuracy, preventing over-reliance on detectors that may occasionally be correct by chance. During the particle filter's weighting stage, trust scores amplify particles that agree with high-trust detectors and dampen those that disagree. If w_i is the base weight of particle i, its trust-adjusted weight is

$$w_i' = w_i \cdot M_i(T_{\text{status}}, T_{\text{red}}, T_{\text{attack}}),$$

Here T_{status}, T_{red}, and T_{attack} denote the evolving trust scores of node status ($\mathcal{D}_{\text{node}}$), Red agent position ($\mathcal{D}_{\text{red}}$) respectively. Each $T \in [0, 1]$ represents the long-term reliability of the corresponding detector. The multiplier $M_i(\cdot)$ combines these values to reward particles that agree with high-trust detectors and down-weight those that conflict, thereby embedding source reliability directly into the belief update process.

5.8 Belief-to-RL Input and Reward Function

To support reinforcement learning in a partially observable setting, the agent must operate on a single state representation at each time step. The particle filtering process begins with a set of unweighted particles sampled from the prior or proposal distribution, representing hypotheses about the true environment state. These particles are then evaluated against incoming observations to compute importance weights, with each particle receiving a weight proportional to the likelihood of the observations given that particle's hypothesized state. This transformation converts an initial uniform particle bag into a weighted particle set that concentrates probability mass on states most consistent with the observed data, producing a belief state representation:

$$b_t = \left\{ (s_t^{(i)}, w_t^{(i)}) \right\}_{i=1}^{N},$$

where each $s_t^{(i)} \in \mathcal{S}$ is a discrete particle and $w_t^{(i)} \in [0, 1]$ is its normalized weight reflecting the posterior probability of that state hypothesis given the observation history.

The goal is to derive a single representative state $\tilde{s}_t$ for use in reinforcement learning (RL) state input and reward computation. Different strategies can be used such as Maximum Weight (MAP), Soft Mode Voting (Fieldwise), Observation-Matching, Top-K Nearest to Aggregated Mode, and Weighted Feature Aggregation [44, 57, 58].

This work uses Weighted Feature Aggregation because of its simplicity, and able to handle high dimensional space. Weighted feature aggregation is used to transform the belief state into a compact vector representation suitable for reinforcement learning. Each particle contributes according to its weight, and features are combined into a single encoding that reflects the agent's current belief distribution. This ensures that the input to the learning algorithm captures both the underlying state information and the uncertainty over possible states, providing a consistent interface between probabilistic reasoning and policy optimisation [36, 59].

5.9 Reinforcement Learning Algorithms

Three RL algorithms were evaluated as Blue agent policies: Proximal Policy Optimization (PPO), Deep Q-Network (DQN), and Advantage Actor-Critic (A2C) [59]. All algorithms receive the aggregated belief state $\tilde{s}_t$ as input and output actions from A_{blue}. The algorithm-specific hyperparameters were selected following reproducible configurations from canonical literature, for example, PPO [38], DQN [37], A2C [60] and validated through small-scale tuning experiments to ensure stable convergence. Configurations of each algorithm can be found in Appendix A

6. EXPERIMENTS PERFORMED AND EVALUATION METRICS

To train, validate, and test our Blue agent architecture, we conducted experiments in a simulated network environment using *CyberBattleSim*, an open-source simulation environment by Microsoft [19]. All experiments were conducted on a Linux virtual machine with 4 CPU cores, 16 GB RAM, and 484.5 GB storage (no GPU) [19]. Two LAN environments were configured: a 7-node LAN and a

14-node LAN, as shown in FIGURE 3 and in FIGURE 12 (Appendix B) respectively. The 7-node LAN comprised four heterogeneous workstations (Windows, Ubuntu, and Linux), one e-commerce web site, one web directory, and a central web server, capturing both diversity of endpoints and high-value services commonly targeted by ransomware [32]. For scalability tests, the 14-node LAN expanded this setup with six workstations, four critical sites (two e-commerce sites and two directories), and four servers (two web servers, an application server, and a database). This configuration reflects the complexity of enterprise networks, with redundant services and databases that enlarge the attack surface and increase the number of attractive targets [61].

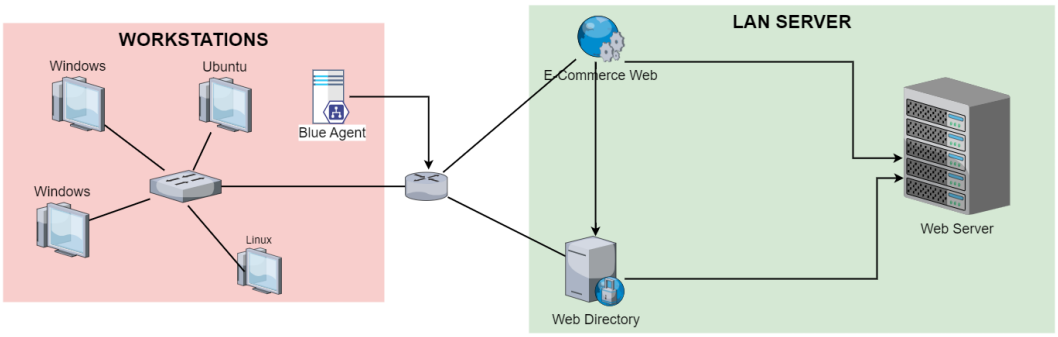

Figure 3: 7-node LAN - One of the workstations is the entry point for the Red agent. The Blue agent does not begin from any node but is instead stationed at a defender-controlled position at the start of the simulation.

6.1 Simulation Environment and Game Flow.

The simulation environment formalizes the cyber defense problem as a turn-based, episodic game between the defender *Blue Agent* and the worm ransomware *Red Agent*. The game proceeds in discrete time steps t until an episode termination condition is met. Both agents maintain probabilistic beliefs over the network state using neural particle filtering.

1. *Red agent action:* The Red agent selects an attack action a_t^{red} based on its particle-filtered belief of the network state.
2. *Environment update:* The environment state S_t transitions to S_{t+1} according to a_t^{red}.
3. *Blue agent observation:* The multi detection system generates observations for the Blue agent. These observations are weighted by the Trust module and fused via particle filtering to produce the belief state B_{t+1}.
4. *Blue agent action selection and safety:* The Blue Agent selects a defensive action a_{t+1}^{blue} based on B_{t+1}. The Shielding module evaluates the action, modifying unsafe actions to safer alternatives a'_{t+1} before execution.
5. *State update and learning:* The environment updates based on executed actions. Both agents update their policies using their respective rewards. The Blue agent's reward may be penalized by the Shielding module, whereas the RL Red agent learns without trust or shielding constraints.

An episode ends if:

1. The Red agent successfully reach the final node (critical server) and exfiltrates or encrypts data from that node.
2. The Blue agent successfully fix all nodes to a Normal condition including removing the Red agent from the network or exhaust Red agent actions.
3. The maximum number of time steps ($T_{\max}$) is reached, also referred to as an Episode timeout.

6.2 RL Red Agent Formulation

We modeled the Red agent under two paradigms: a rule-based adversary following scripted attack strategies with predefined progression paths, and an adaptive learning adversary trained via reinforcement learning (PPO) that dynamically adjusts its strategy based on Blue's defenses.To understand how the red agent process its information, we created a scan function as its detection model. In contrast to the Blue agent, which uses detectors to derive observations, the Red agent gathers information through an active scan function $\mathcal{S}_{\text{red}}$. This function probes the environment and returns a structured observation vector:

$$O_{\text{red}} = \{A_{\text{curr}}, B_{\text{tgt}}, C_{\text{currStat}}, D_{\text{tgtStat}}, E_{\text{neigh}}, F_{\text{creds}}, G_{\text{neighVul}}, H_{\text{currVul}}\}$$

where:

- A_{curr}: current Red agent position,
- B_{tgt}: target node under attack,
- C_{currStat}: scanned status of the current node,
- D_{tgtStat}: scanned status of the target node,
- E_{neigh}: list of discovered neighbour nodes,
- F_{creds}: set of discovered credentials,
- G_{neighVul}: vulnerabilities of neighbour nodes,
- H_{currVul}: vulnerabilities of the current node.

Scanned status of the current node, C_{currStat}, and scanned status of the target node, D_{tgtStat}, can be one of the following:

- `red_installed`: the Red agent is installed in the node,
- `encryption`: the node's files are encrypted,
- `owned`: the node is fully controlled via privilege escalation,

- `stopped`: the node is inactive or unavailable,
- `running`: the node is operational and unaffected.

To model uncertainty, $\mathcal{S}_{\text{red}}$ applies rule-based scoring to each attribute, converting the evidence into probabilities with the softmax function (Eq. 6). The final observation is thus a probabilistic estimate of the environment from the attacker's perspective, reflecting noisy reconnaissance rather than ground truth. The RL Red agent is formalised as a POMDP, introduced earlier, just as the Blue agent. The RL Red agent's action space is

$$A_{\text{red}} = \left\{ \begin{array}{l} \texttt{scan}, \\ \texttt{install\_self}, \\ \texttt{local\_privilege\_escalation}, \\ \texttt{remote\_privilege\_escalation}, \\ \texttt{lateral\_move}, \\ \texttt{file\_encryption}, \\ \texttt{local\_sql\_injection}, \\ \texttt{remote\_sql\_injection}, \\ \texttt{denial\_of\_service}, \\ \texttt{hide\_self}, \\ \texttt{do\_nothing} \end{array} \right\}.$$

The RL Red agent observation is:

$$O_{\text{red}} = \left\{ \begin{array}{l} A_{\text{curr}} \text{ (current Red agent position)}, \\ B_{\text{tgt}} \text{ (target node)}, \\ C_{\text{currStat}} \text{ (scanned status of current node)}, \\ D_{\text{tgtStat}} \text{ (scanned status of target node)}, \\ E_{\text{neigh}} \text{ (discovered neighbour nodes)}, \\ F_{\text{creds}} \text{ (discovered credentials)}, \\ G_{\text{neighVul}} \text{ (vulnerabilities of neighbour nodes)}, \\ H_{\text{currVul}} \text{ (vulnerabilities of current node)} \end{array} \right\}.$$

To enable symmetric partial observability, the RL Red agent implements the same belief-tracking architecture as the Blue agent, using transition and observation models with particle filtering to maintain probabilistic beliefs. In a simplified way, the Red agent is rewarded for spreading through the network, escalating privileges, and successfully reaching high-value targets, for example the Web Server in the 7-node LAN or the Database Server in the 14-node LAN. Terminal rewards are assigned when either agent achieves its win condition.

6.3 Experiments and Results

TABLE 2 and TABLE 3 show the experiments that were conducted and evaluation metrics respectively. Our experiments are organized into five phases: *Algorithm Selection*, *Detection Scheme Evaluation*, *Scalability*, *Validation*, and *Benchmarking*. All experiments were conducted with 5

Table 2: Summary of experiments performed.

Setting	Blue Agent Variants	Opponent
7 nodes	PPO, DQN, A2C (baseline)	Rule-based Red
	PPO, DQN, A2C (trust only)	Rule-based Red
	PPO, DQN, A2C (shield only)	Rule-based Red
	PPO, DQN, A2C (shield + trust)	Rule-based Red
	PPO, DQN, A2C (single detector)	Rule-based Red
14 nodes (scalability)	Best Blue from 7-node (shield + trust)	Rule-based Red
14 nodes (adversarial)	Best Blue from 7-node (shield + trust)	RL Red (PPO)
Benchmark comparison	PPO (DLR-RM/stable-baselines3)	Rule-based Red

Table 3: Evaluation metrics with formulas and interpretations.

Metric	Formula	Meaning
Win rate	$W = \frac{N_{\text{blue wins}}}{N_{\text{episodes}}}$	Percentage of episodes where the Blue agent succeeds [37, 59].
Availability loss	$L_{\text{avail}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{1}\big[a_t = \text{isolate}\big]$	Fraction of steps where nodes are unavailable due to isolations across episodes [62].
Data loss	$L_{\text{data}} = \sum_{t=1}^{T} \mathbf{1}\big[a_t = \text{reset}\big]$	Count of unsafe resets on nodes across episodes [62].
Average return	$\bar{R} = \frac{1}{N}\sum_{i=1}^{N} R_i$	Mean cumulative return per episode [59].
Trust score	$T_{t+1} = T_t + \beta T_t(1 - T_t)\, c_t$	Confidence in detectors, updated via logistic growth based on correctness feedback [50, 51].
KL divergence	$D_{\text{KL}}(P \parallel Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$	Measures divergence between probability distributions; lower values imply higher policy stability [63].

independent runs using different random seeds (42, 123, 456, 789, 1000) to ensure reliability. To formally validate the performance differences, a one-way Analysis of Variance (ANOVA) and Mann-Whitney U test was performed across algorithm groups, followed by post-hoc Tukey's Honestly Significant Difference (HSD) tests for pairwise comparisons [64]. Unless otherwise specified, all reported performance differences are statistically significant at the $p < 0.05$ level.

6.3.1 Experiment A: Detection scheme evaluation

To empirically validate the contribution of *Multi detection system* against prior work's reliance on single detection system, we conducted a comparative analysis across three core algorithms: PPO, DQN, and A2C on a 7-node environment against rule based Red agent. Each algorithm was trained

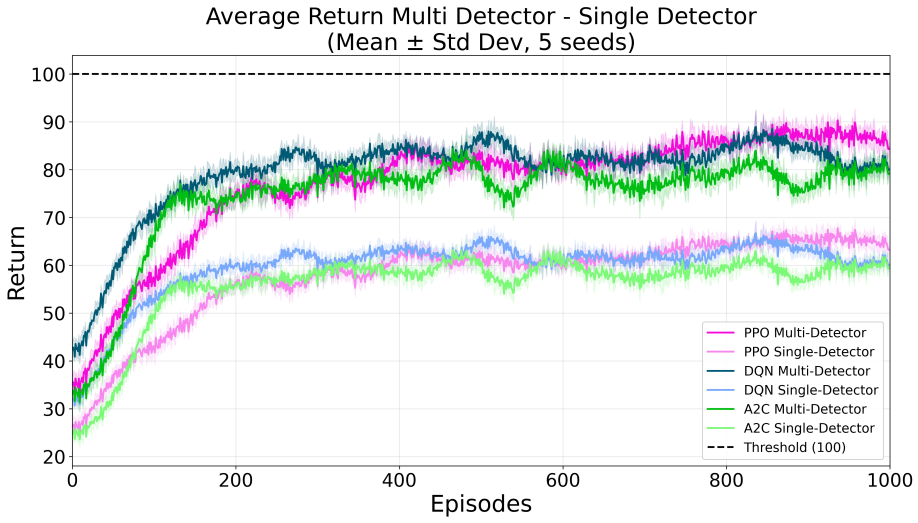

Figure 4: PPO, DQN, and A2C. Each algorithm was trained and evaluated independently under two distinct observation settings

and evaluated independently under two distinct detection settings: 1) The novel *Multi detection system*, which uses both network and node detector outputs into observation, and 2) a traditional *Single detection system*, which represents the standard observation space found in the existing literature [12, 30, 55, 56] and modeled at 5.2. We tracked only average returns for this experiment to see how each algorithm will perform in different detection settings. The performance comparison illustrated in FIGURE 4 demonstrates that the Multi-detectors configuration achieves a superior and near-optimal policy convergence across all tested algorithms. Specifically, the Multi-detectors PPO agent converged to a stable mean expected return of 82.15 ± 1.98, representing a substantially more optimal policy than the Single-detector PPO, which stagnated at a return of 61.61 ± 1.48. A similar trend was observed for DQN, where the Multi-detectors agent exhibited stronger policy learning, converging to an average return of 84.17 ± 1.79, a marked improvement over the Single-detector DQN's suboptimal convergence to 62.63 ± 1.34. Furthermore, the Multi-detectors A2C agent demonstrated superior performance stability and convergence, reaching 78.36 ± 1.96 compared to the Single-detector agent's 58.77 ± 1.47 ($ANOVA$, $p < 0.001$).

6.3.2 Experiment b: Selection phase

This phase evaluated PPO, DQN, and A2C under four configurations against a rule-based Red agent in a 7-node environment.

Focus on PPO: Across all metrics, the PPO algorithm demonstrated superior stability and final performance. We focus on PPO figures in the main text, and complete figures for DQN and A2C are provided in the supplementary material (`https://github.com/VinceTaku/RL_training_experiment_results`).

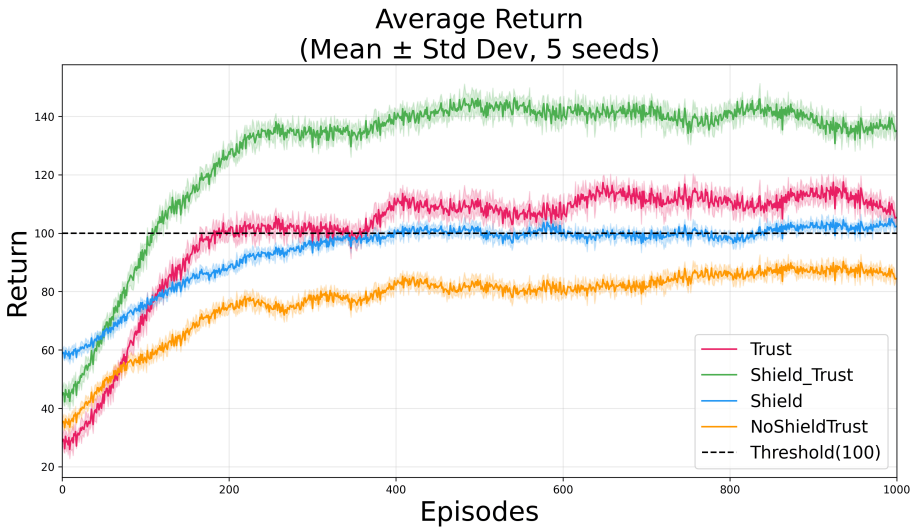

Figure 5: PPO average return over training episodes under different configurations (baseline, trust only, shield only, shield + trust). Solid lines show mean across 5 random seeds; shaded regions indicate ±1 standard deviation.

Average Returns (Policy Performance). FIGURE 5 illustrates the PPO policy learning curves. The dashed black line, representing a return of 100, signifies the environment survival threshold or the minimum expected return required for a successful defense policy [25].

The combined Shield and Trust configuration achieved the most optimal policy, resulting in the highest stable expected return of 141.97 ± 2.82 and significantly exceeding the survival threshold. The Shielding-only policy also demonstrated robust convergence and achieved a successful expected return beyond the threshold, stabilizing at 101.69 ± 1.72, and the Trust-only policy (110.68 ± 2.90). Conversely, the baseline policy (82.15 ± 1.98) failed to maintain an expected return above the critical 100 threshold. Statistical analysis confirmed that PPO (Shield and Trust) achieved a significantly higher mean return compared to all other configurations tested, including the best-performing DQN configuration (Shield and Trust)(120.58 ± 2.52) and A2C configuration (Shield and Trust) (110.57 ± 2.50) (Tukey's HSD, $p < 0.001$), which are not shown in the main plot but are in the supplement material (`https://github.com/VinceTaku/RL_training_experiment_results`).

Trust Dynamics. FIGURE 6 displays the evolution of detector trust scores. The Red Position detector demonstrated the fastest convergence (plateauing at 0.98 ± 0.02 by episode 150). The Status Detector followed closely (0.97 ± 0.03 around episode 200). The *Attack Target detector* exhibited the slowest initial learning rate and delayed stabilisation at 0.96 ± 0.03. This delay is attributed to the inherent characteristics of the attack event itself. Consequently, the detector requires a longer period of cumulative observation and learning to achieve better generalisation and confidence (trust) compared to detectors tracking frequent state changes. All detectors converged to reliable trust values by episode 300.

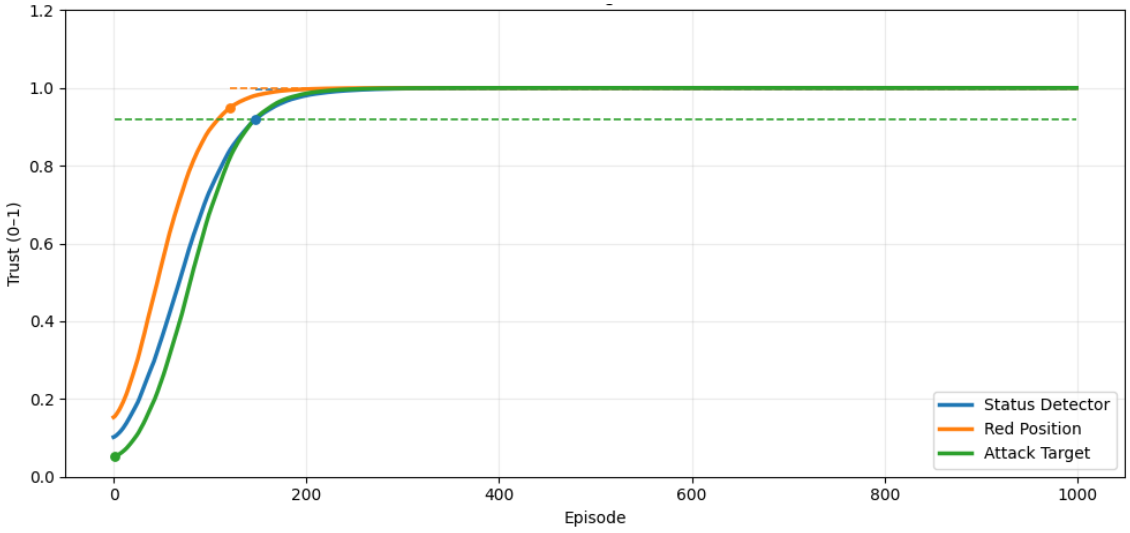

Figure 6: Trust score evolution across training episodes for three detectors. Lines show mean trust values; across 5 runs.

Safety Metrics. FIGURE 7 reports PPO's data-loss resets and availability loss, demonstrating the effect of the *Probabilistic Shielding* mechanism. The shielded PPO configurations (Shield and ShieldTrust) show clear superiority, with both availability loss and data loss dropping to near-zero ($0.2 - 0.3\%$) by episode 200. In contrast, configurations without the shield persist at high data-loss rates (around $5.0 \pm 0.6\%$) and elevated availability loss (approximately 1% and 2% for Trust-only and NoShieldTrust, respectively).

DQN and A2C exhibit algorithm-specific safety behaviors (full results in supplementary materials (`https://github.com/VinceTaku/RL_training_experiment_results`)). For availability loss, non-shielded configurations remain unstable: A2C Trust-only fluctuates between $3.0{-}3.5\%$ by episode 200 after an initial drop from 8%, while A2C NoShieldTrust converges to $1.0 - 1.8\%$ following a decrease from 7%. Shielded A2C configurations perform better: Shield-only converges to $0.8 - 1.0\%$, and ShieldTrust to $2.0 - 3.0\%$. For data loss, shielded configurations were better, achieving $2.0 - 2.5\%$ and $< 1\%$ for ShieldTrust and Shield only respectively.

For DQN, availability loss converges across all configurations to $0.2 - 0.5\%$ after a significant drop from 6%, comparable to PPO's shielded configurations ($p < 0.01$). Data loss for DQN shows minimal improvement: all configurations stabilize around $4.0 - 4.5\%$, with shielded configurations slightly increasing from 3%, while non-shielded configurations remain unchanged throughout training. Overall, PPO demonstrates the strongest shield response, highlighting the pronounced impact of Probabilistic Shielding.

Win Rates. TABLE 4 summarises the final win rates. The PPO with Shield and Trust configuration achieved the highest win rate at $93 \pm 2\%$, showing the results were very consistent across the independent training runs and representing a statistically significant improvement over both other PPO's configurations and other algorithms (Mann-Whitney U test, $p < 0.01$ [64]).

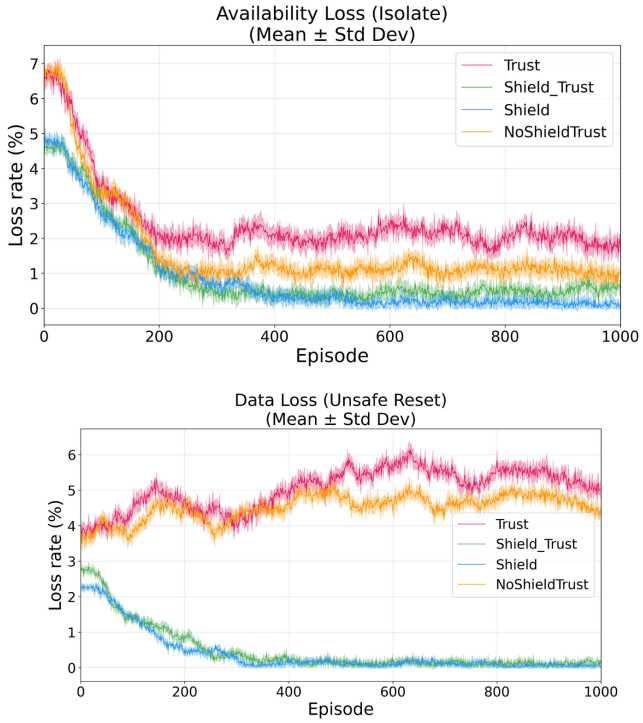

Figure 7: Data-loss resets and availability loss for PPO under different configurations against a rule-based Red agent in a 7-node environment. Lines show mean loss percentages; shaded regions indicate ±1 standard deviation across 5 runs.

Table 4: Win rates (%) of Blue agent algorithms against rule-based Red agent in 7-node environment. Values show mean ± standard deviation over 5 runs.

Algorithm	**No S&T**	**Trust only**	**Shield only**	**Shield & Trust**
PPO	62 ± 3	57 ± 5	89 ± 2	$\mathbf{93 \pm 2}$
DQN	66 ± 6	69 ± 5	67 ± 4	70 ± 5
A2C	64 ± 5	52 ± 7	79 ± 4	66 ± 6

Mean KL-Divergence. The KL-divergence results in Table 5 provide a measure of policy stability across algorithms and configurations, with lower values indicating more consistent action distributions over time. PPO configurations with Shield and Trust achieve the most stable policies, exhibiting the lowest KL values and minimal variance. A2C configurations show moderate stability, with Shield+Trust improving over Shield-only. DQN configurations demonstrate the least stable policies, with Shield+Trust performing slightly worse than Shield alone.

Table 5: Mean KL-Divergence (± std) across algorithms and configurations, ranked by policy stability (lower is better).

Rank	Algorithm	Mean KL	Std. Dev.
1 (Best)	**PPO + Shield + Trust**	**0.010**	**0.001**
2	PPO + Shield	0.013	0.002
3	A2C + Trust + Shield	0.021	0.004
4	A2C + Shield	0.039	0.005
5	DQN + Shield	0.055	0.007
6 (Worst)	DQN + Shield + Trust	0.058	0.008

6.3.3 Experiments C and D: Scalability and validation phase (14-node network)

This phase tested the PPO (Shield and Trust) model's generalisation to a larger, more complex 14-node network against both rule-based and RL-trained Red agents.

Training Time. Training time scaled from 10.2 ± 0.8 hours (7-node) to 24.8 ± 1.5 hours (14-node) against the rule-based adversary, approximately $2.4\times$ longer. Training against the RL-trained adversary required 43.5 ± 2.8 hours, an increase of approximately $1.75\times$ over the rule-based adversary on the same network size.

Average Returns and Win Rate (Policy Performance). FIGURE 8, shows the average return comparing two opponent policy strategies on the 14-node network. Against the rule-based Red policy (green curve), the Blue agent demonstrates rapid policy learning, consistently exceeding the 100 expected return threshold by episode 100 and achieving stable expected returns of 115.56 ± 1.35 in the final 400 episodes with a win rate of $95 \pm 2\%$. Conversely, against the adaptive, RL-trained Red policy (blue curve), the Blue agent's policy convergence is slower, reaching the threshold by episode 150 and stabilizing around 105.23 ± 1.38 in the final 400 episodes. The rule based opponent yields statistically superior overall performance compared to the adaptive RL opponent policy (Mann-Whitney U test, $p < 0.01$). The RL Red agent's policy exhibits significantly higher variance between episodes 200–600, with periodic performance oscillations. Despite these differences, the Blue agent's defense policy in both scenarios converges above 105 return in later episodes, maintaining performance above the success threshold, and achieving a $95 \pm 2\%$ win rate against the fixed opponent and a $73 \pm 3\%$ win rate against the adaptive, learning-based opponent.

Data Loss and Availability Loss. FIGURE 9 and 10, show the safety metrics across both opponent types. For availability loss, both scenarios show similar initial decline, stabilizing around $0.5 \pm 0.3\%$ (rule-based) and $1.2 \pm 0.8\%$ (RL-trained) by episode 200. The RL scenario exhibits more persistent sporadic spikes throughout training. For data loss, the rule-based scenario reaches $0.4 \pm 0.3\%$ by episode 200 and remains minimal thereafter. In contrast, the RL scenario shows a gradual increase after episode 400, stabilizing around $2.3 \pm 0.5\%$ for episodes 600–1000, compared to $0.2 \pm 0.2\%$ in the rule-based scenario.

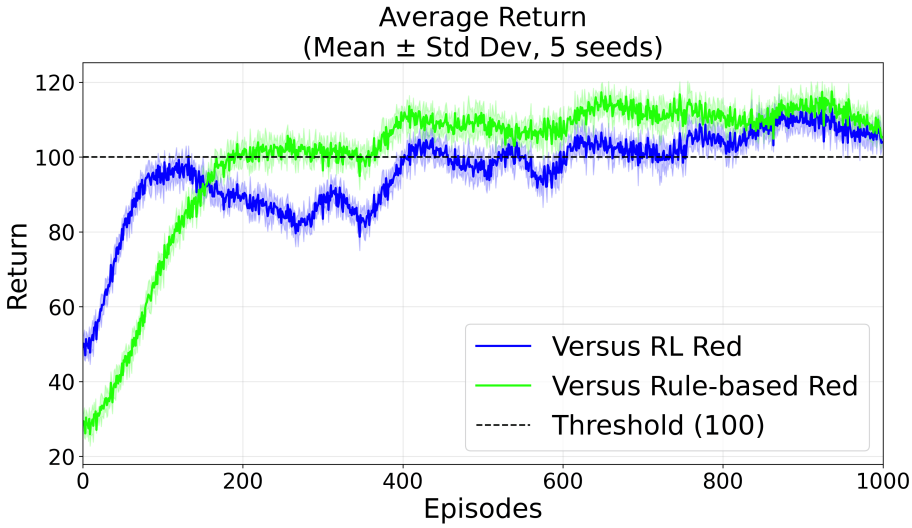

Figure 8: Blue agents with shield and trust average return comparison between rule-based and RL-trained red opponents on the 14-node network (Mean ± Std Dev, 5 seeds).

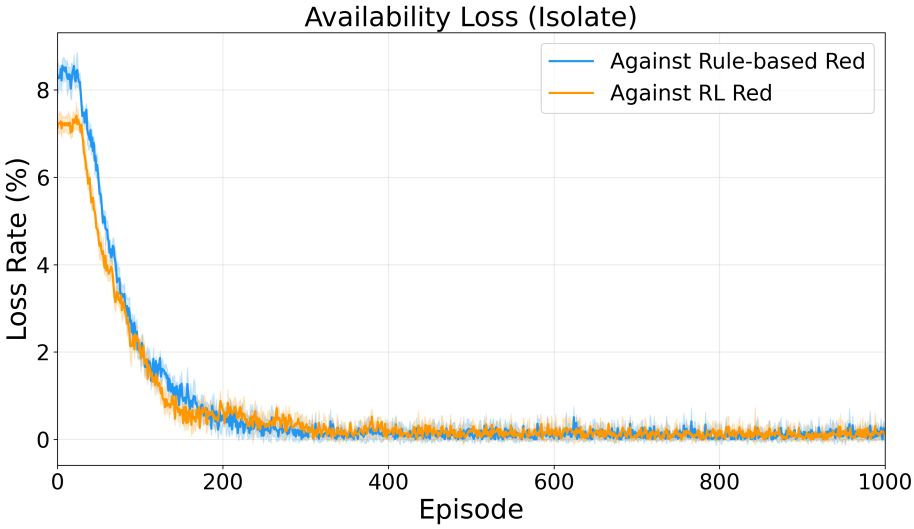

Figure 9: Availability loss comparison for PPO with shield and trust against rule-based and RL-trained Red agents in 14-node environment. Lines show mean loss percentages; shaded regions indicate ±1 standard deviation across 5 runs.

6.3.4 Experiment e: Benchmarking

We compared our approach to a standard PPO implementation (DLR-RM/stable-baselines3) [56], in the 7-node environment against rule-based Red agent. The standard PPO benchmark was trained using the identical observation space, belief representation, network architecture, and reward function

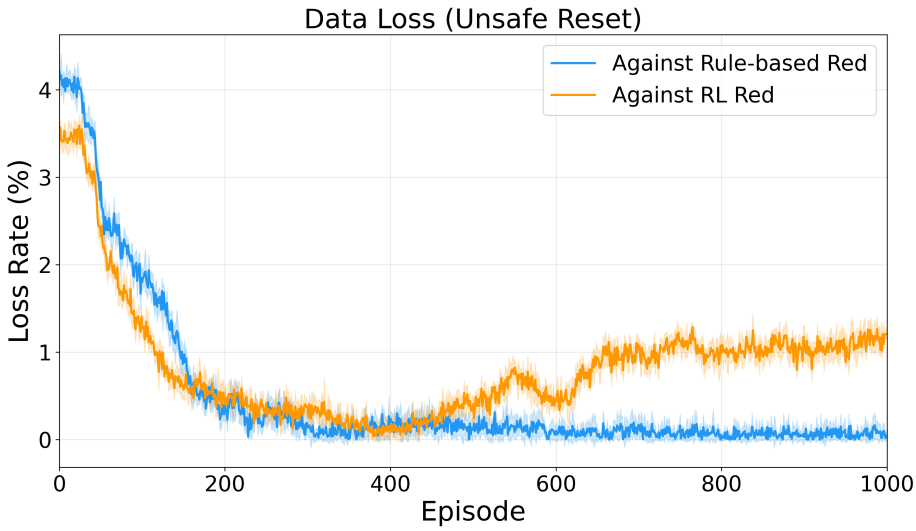

Figure 10: Data-loss resets comparison for PPO with shield and trust against rule-based and RL-trained Red agents in 14-node environment. Lines show mean loss percentages; shaded regions indicate ±1 standard deviation across 5 runs.

as our full PPO (Shield and Trust) model. This methodology ensures that the observed performance differences are isolated and directly attributable to the presence or absence of our proposed Trust and Shield mechanisms. FIGURE 11, shows the benchmark agent's learning progression. The benchmark agent showed highly unstable performance, with returns declining progressively after episode 400 and stabilising around average return of 48.87 ± 2.75. The final win rate was $32 \pm 1\%$.

6.3.5 Training Performance and Profiling

Training times were: 7-node environment (10.2 ± 0.8 hours), 14-node environment (24.8 ± 1.5 hours), and 14-node with learning adversary (43.5 ± 2.8 hours) for 1000 episodes. The 2.4 times scaling from 7 to 14 nodes reflects state space growth $O(|S_{\text{node}}|^K)$ and quadratic connection complexity, while the 1.75 times increase for learning adversaries reflects co-adaptation dynamics.

Profiling reveals episodic neural network training accounts for 60% of total time (BeliefNet: 35%, StateNet: 25%, RL updates: 40%), with particle filtering at 30% and trust/shielding negligible (<1%).

7. DISCUSSION

This section interprets our experimental results in relation to identified research gaps, and research questions mentioned earlier.

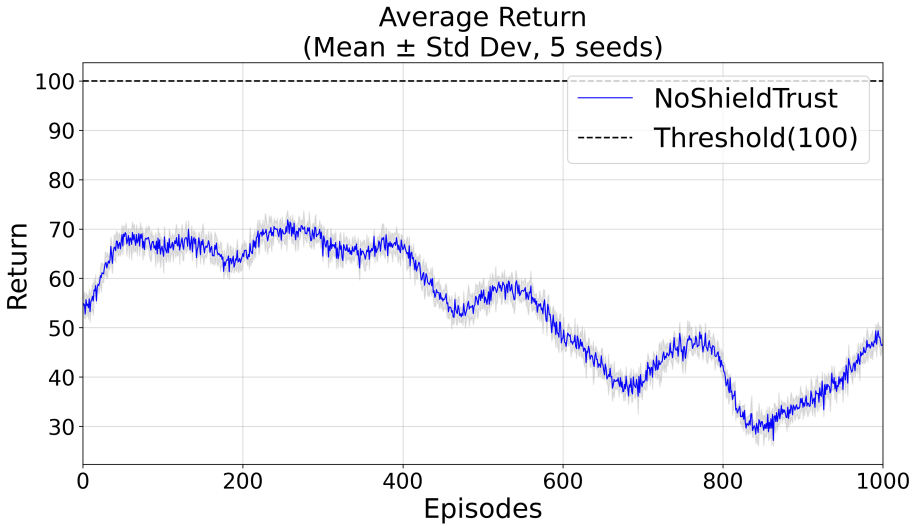

Figure 11: Average return of the benchmark PPO algorithm against rule-based Red agent in 7-node environment. Line shows mean across 5 runs; shaded region indicates ±1 standard deviation.

7.1 Multi Detectors System

Experiment A's consistent average return improvement across all algorithms (PPO, DQN, A2C) when using multi detection versus single detection reveals a fundamental limitation in prior work [18, 25]. Single detectors proved to create information bottlenecks that may affect agent learning capacity. Network detector observe traffic flows but miss local node status, node detector see node status but miss attacker movement patterns. Neither alone provides sufficient context for the agent to understand spreading attacks that exploit lateral movement, precisely the threat model worm ransomware represents. With the multi-detection system, although each detector may be imperfect, their combined use provides broader observation coverage across nodes and attributes, reduces the variance in state estimation, and enhances the consistency of belief updates over time. This improved state awareness allows the Blue agent to form a more close-to-accurate understanding of the network's state, leading to more stable policy gradients and more reliable learning regardless of the underlying learning algorithm. This answers the research question, *"How does a multi-detector system impact an RL agent's learning performance compared to a single-detector system in partially observable network defense scenarios?"*

7.2 Impact of Probabilistic Shielding

Probabilistic shielding affects each algorithm differently by controlling exploration away from unsafe actions. Without shielding, agents exploit these actions for short-term rewards, creating unsafe policies that ignore better long-term strategies. PPO benefits most from shielding, achieving the highest performance (141.97 ± 2.82 return, 93% win rate) with minimal safety violations. Low KL-divergence (0.010) confirms stable policy updates. A2C shows mixed results. It effectively

reducing data loss but struggling with availability loss due to high-variance gradients and limited exploration under constraints. DQN reduces availability loss but fails to prevent unsafe resets due to off-policy learning and uniform replay buffer sampling that ignores shielded probabilities.

From the results, three conditions we concluded that enable effective shielding: (i) sufficient exploration diversity, (ii) stable policy gradients, and (iii) on-policy or prioritized learning that respects shielded probabilities. PPO satisfies all three; A2C and DQN do not. Against the learning Red agent (Experiment D), PPO's small increase in data loss (0.4% to 1.1%) while maintaining 73% win rate demonstrates adaptive safety meaning the shield dynamically adjusts constraints based on threat level without compromising defensive effectiveness. This addresses the question, *"How well does probabilistic shielding help reinforcement learning agents explore safely and prevent data or service loss without hurting their performance?"*

7.3 Impact of Trust

Trust alone improves short-term returns but not win rates. For PPO, trust increases return from 82.15 to 110.68 but decreases win rate from 62% to 57%. Trust improves state estimation by weighting reliable detectors more heavily, enabling better short-term decisions. However, without shielding, agents execute unsafe strategies more confidently rather than learning safer ones. Combining Trust and Shield proved to be more beneficial as PPO achieves 141.97 return and 93% win rate (versus 101.69 and 89% with shielding alone). Shielding constrains exploration to safe regions; trust improves decision accuracy within those regions. Trust proves most valuable by preventing overreliance on uncertain observations from detectors.

To answer this research question, *"How do trust mechanisms improve agent learning under noisy detector observations, and how does trust interact with shielding to enhance policy safety and effectiveness?"*, we conclude that trust helps the agent learn better under noisy detector observations by giving more weight to accurate detectors and reducing the impact of false readings. This improves state estimation and makes learning more stable. However, trust alone does not make the agent safer, it only improves perception. When combined with probabilistic shielding, trust enhances both safety and performance: shielding keeps the agent's actions within safe limits, while trust ensures accurate decisions within that safe space. Together, they lead to safer and more effective policy learning.

7.4 Blue Agent Against Rule-Based and Learning Red Agent

Experiments C and D show how the Blue agent performs against two different types of Red agents. Against the rule-based Red agent, which follows fixed attack patterns, the Blue agent learns faster and performs better because the opponent's actions are predictable. This stable environment helps the Blue agent quickly find effective defense strategies. In contrast, the RL-trained Red agent is more challenging because it learns and changes its behavior over time. This makes the Blue agent's learning less stable, with more variation in performance, since both sides keep adapting to each other. Although the rule-based Red agent is useful for early training and testing, the learning Red agent provides a more realistic and difficult test of the Blue agent's ability to adapt and remain effective. The Blue agent still keeps its performance above the survival threshold in both cases, showing strong resilience. However, the higher variation against the learning Red agent shows that perfect stability

is hard to achieve in adaptive, adversarial environments. Finally, the Blue agent keeps availability loss near zero against both Red agents, showing the shield's reliability in controlling isolations. The small increase in data loss after episode 400 reflects an ongoing battle when the Red agent learns new attack patterns, the Blue agent must adjust its safety limits to respond.

7.5 Benchmark PPO versus PPO RL with Shield and Trust

Experiment E's benchmark poor performance (achieving only 48.87 ± 2.75 average return and $32 \pm 1\%$ win rate) versus our PPO with Shield and Trust configuration reveals two compounding failure modes, rooted in a fundamental lack of generalization. The standard PPO agent, optimized for a simpler training environment, fails to generalize effectively to the conditions of our more complex simulation. The poor performance specifically after episode 400 suggests policy collapse, indicating the agent converged to a fundamentally unstable strategy that actively worsens outcomes when faced with complexity. Our Shield and Trust configuration, by contrast, shows stable or improving performance because shielding prevents convergence to destabilizing policies and trust maintains belief integrity under observation noise.

7.6 Scalability and Generalization

When scaling to 14 nodes, whether trained against rule-based or learning Red agents, the Blue agent maintained average returns above the threshold and reasonably high win rates, highlighting the strong of belief maintenance. Sustained performance despite a $2.4\times$ increase in training time indicates that the neural observation and transition models generalize effectively, capturing underlying patterns in the environment rather than memorizing specific network configurations.

7.7 Comparison of Transformer-based RL agents and our approach

Our method differs from transformer-based RL agents such as Decision Transformer [65], Trajectory Transformer [66], and Reinformer as summarized in Table 6. Transformers treat trajectories as sequences and predict the next action using attention. They require large offline datasets and do not explicitly use trust, shielding, or multiple detectors. Our approach uses belief-based inputs from multiple detectors and updates policies online. We incorporate trust weighting, probabilistic logic shielding, and multi-detector coordination. This allows fast adaptation to attacks, interpretable decisions, and safe actions in LAN defense environments. Compared to transformers, our method focuses on domain-specific performance and safety. However, our method requires careful tuning, scales less easily to very large networks, and may underperform when trajectories are long or highly stochastic.

8. CONCLUSION

This research tackled major limitations in previous RL-based cyber defense systems by combining three key mechanisms: multi detection system, trust system, and probabilistic shielding. Exper-

Table 6: Comparison of Transformer-based RL agents and our approach.

Study / Approach	Model Type	Learning Setting	Environment	Notable Features
[65] (Decision Transformer)	Transformer policy	Offline RL	MuJoCo, Atari	Predicts next action from past trajectory + target return
[66] (Multimodal Transformer RL)	Multimodal Transformer	Offline RL	Simulated RL tasks	Separate embeddings for states, actions, rewards; improved offline performance
(Reinformer)	Transformer sequence model	Offline RL	Standard benchmarks (Gym)	Max-return sequence modeling; trajectory stitching for offline RL
Ours	PPO + structured policy/value networks	Online RL	LAN / CyberBattleSim	Multi-detector input, trust weighting, shielding, policy shaped by node-level observations

iments using PPO, DQN, and A2C in both 7-node and 14-node LANs showed the value of each component. Multi detection system, which combines network-level and node-level observations, consistently improved average returns for all algorithms, showing that having a complete view of the network is essential for effective learning. Probabilistic shielding helped agents avoid unsafe actions during training, reducing both data and availability losses while improving win rates. Trust-based belief updates improved the accuracy of state estimates by weighting detector inputs based on their historical reliability. Together, these mechanisms produced an RL agent that is aware and safe against worm ransomware. The system scaled successfully to 14-node networks and outperformed standard PPO baselines while keeping training times reasonable (10–44 hours on CPU-only machines). This demonstrates progress toward practical, RL-based cyber defense.

However, some limitations remain. All tests were in simulation, so real-world factors like unexpected attack types, detector failures, or long-term changes are not fully captured. Training still takes a long time, limiting fast adaptation to new attacks. Also, the current single-agent setup may not scale well to very large networks that need decentralized decision-making.

As part of future work, we plan to validate our approach on real-world datasets such as CICIDS2017 [45] and NSL-KDD. To improve efficiency, we aim to implement GPU-parallelized particle filtering to accelerate training and develop incremental belief update algorithms that allow online adaptation to emerging attack patterns. Scalability will be addressed through a hierarchical multi-agent system, where local defenders manage network segments under a centralized coordinator and have a cooperative defense. Finally, we will explore human-in-the-loop integration, designing interfaces for security analysts to provide corrective feedback during operation.

References

[1] Sirkemaa S, Suomi R. Towards the Knowledge Society. towards the knowledge society: eCommerce eBusiness and eGovernment The Second IFIP Conference on E-Commerce E-Business E-Government. J E-Gov. 2003;I3E:463-477.

[2] Oz H, Aris A, Levi A, Uluagac AS. A Survey on Ransomware: Evolution Taxonomy and Defense Solutions. ACM Comput Surv. 2022;54:1-37.

[3] Chen Q, Islam SR, Haswell H, Bridges RA. Automated Ransomware Behavior Analysis: Pattern Extraction and Early Detection. In International Conference on Science of Cyber Security. Cham: Springer International Publishing. Springer Nature. 2019:199-214.

[4] Wang C, Redino C, Clark R, Rahman A, Aguinaga S, et al. Leveraging Reinforcement Learning in Red Teaming for Advanced Ransomware Attack Simulations. In 2024 IEEE International Conference on Cyber Security and Resilience. CSR. IEEE. 2024:262-269.

[5] https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024

[6] https://www.verizon.com/business/resources/Tc02/reports/2025-dbir-data-breach-investigations-report.pdf?msockid=0cc8159c2a3a6de23d6103362b0f6ceb.

[7] https://www.bakerdonelson.com/webfiles/Publications/20250822C ost − of − a − Data −Breach − Report−2025pdf.

[8] https://www.netscout.com/blog/ddos-next-generation.

[9] M. Kiely, D. Bowman, M. Standen, C. Moir. On Autonomous Agents in a Cyber Defence Environment. 2023. arXiv preprint: https://arxiv.org/pdf/2309.07388.

[10] M. O. Farooq, T. Kunz. Combining Supervised and Reinforcement Learning to Build a Generic Defensive Cyber Agent. Journal of Cybersecurity and Privacy. 2025;5:23.

[11] A. Dutta, S. Chatterjee, A. Bhattacharya, M. Halappanavar. Deep Reinforcement Learning for Cyber System Defense Under Dynamic Adversarial Uncertainties. 2023. arXiv preprint https://arxiv.org/pdf/2302.01595.

[12] V. Banda, D. Blaauw, B. W. Watson. Towards a Supervised Machine Learning Algorithm for Cyberattacks Detection and Prevention in a Smart Grid Cybersecurity System. In Pan African Conference on Artificial Intelligence. Springer Naturer. 2023:107-128.

[13] M. C. Ghanem, T. M. Chen, and E. G. Nepomuceno. Hierarchical Reinforcement Learning for Efficient and Effective Automated Penetration Testing of Large Networks. Journal of Intelligent In- formation Systems. 2023;60:281-303.

[14] J. Chen, S. Hu, H. Zheng, C. Xing, G. Zhang. GAIL-PT: An Intelligent Penetration Testing Framework With Generative Adversarial Imitation Learning. Computers Security. 2023;126:103055.

[15] Y. Li, H. Dai, J. Yan. Knowledge-Informed Auto-Penetration Testing Based on Reinforcement Learning With Reward Machine. 2024. arXiv preprint: https://arxiv.org/pdf/2405.15908.

[16] R. Kerr, S. Ding, L. Li, A. Taylor. Accelerating Autonomous Cyber Operations: A Symbolic Logic Planner Guided Reinforcement Learning Approach. 2024 International Conference on Computing Networking and Communications. ICNC. IEEE. 2024:641-647.

[17] T. Cody, E. Meno, D. Jones, T. Erpek, P. A. Beling. Lessons Learned in Designing a Cyber Reinforcement Learning Competition. Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications VI. SPIE. 2024;13051:397-407.

[18] J. Huang, Q. Zhu. PenHeal: A Two-Stage LLM Framework for Automated Pentesting and Optimal Remediation. Proceedings of the Workshop on Autonomous Cybersecurity. 2023:11-22.

[19] https://www.microsoft.com/en-us/research/project/cyberbattlesim/

[20] M. Standen, M. Lucas, D. Bowman, T. J. Richer, J. Kim, et al. CybORG: A Gym for the Development of Autonomous Cyber Agents. 2021. arXiv preprint: https://arxiv.org/pdf/2108.09118.

[21] D. Schabinger, C. Bierbrauer, M. Ahner. Dynamic Reinforcement Learning for Network Defense: Botnet Detection and Eradication. Proceedings of SPIE Defense + Commercial Sensing. SPIE. 2024;13051:419-429.

[22] E. Miehling, M. Rasouli, D. Teneketzis. A POMDP Approach to the Dynamic Defense of Large-Scale Cyber Networks. IEEE Transactions on Information Forensics and Security. 2018;13:2490–2505.

[23] https://www.fnc.co.uk/media/mwcnckij/us24milesfarmerreinforcementlearningforautonomousresilientcyberdefen wp.pdf.

[24] https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/2024/7966713

[25] G. Palmer, C. Parry, D. J. B. Harrold, C. Willis. Deep Reinforcement Learning for Autonomous Cyber Defence: A Survey. 2023. arXiv preprint: https://arxiv.org/pdf/2310.07745v1.

[26] S. Rose, O. Borchert, S. Connelly, and S. Mitchell, “Zero Trust Architecture,” National Institute of Standards and Technology (NIST), SP 800-207, 2020.

[27] S. Herse, J. Vitale, M. A. Williams. Simulation Evidence of Trust Calibration: Using Pomdp With Signal Detection Theory to Adapt Agent Features for Optimised Task Outcome During Human-Agent Collaboration. International Journal of Social Robotics. Springer Nature. 2024;16:1381–1403.

[28] H. L. Fung, V. A. Darvariu, S. Hailes, M. Musolesi. Trust-Based Consensus in Multi-Agent Reinforcement Learning Systems. 2022. arXiv preprint arXiv: https://arxiv.org/pdf/2205.12880v1.

[29] A. KazemiNajafabadi and M. Imani. Optimal Joint Defense and Monitoring for Networks Security Under Uncertainty: A POMDP-Based Approach. IET Information Security. 2024;2024:7966713.

[30] Y. Zhang, W. Cai, H. Chen, Z. Qi, H. Chen, et al. A Learning-Based Pomdp Approach for Adaptive Cyber Defense Against Multi-Stage Attacks. Proceedings of the 26th IEEE International Conference on High Performance Computing and Communications. HPCC. IEEE. 2024:1220-1227.

[31] A. Dutta, S. Chatterjee, A. Bhattacharya, M. Halappanavar. Deep Reinforcement Learning for Cyber System Defense Under Dynamic Adversarial Uncertainties. 2023. arXiv preprint: https://arxiv.org/pdf/2302.01595.

[32] Y. Tang, J. Sun, H. Wang, J. Deng, L. Tong, et al. A Method of Network Attack–Defense Game and Collaborative Defense Decision-Making Based on Hierarchical Multi-Agent Reinforcement Learning. Computers Security. 2024;142:103871.

[33] E. Khalastchi, M. Kalech. Fault Detection and Diagnosis in Multi-Robot Systems: A Survey. Sensors. 2019;19:4019.

[34] J. Sabater, C. Sierra. Review on Computational Trust and Reputation Models. Artificial Intelligence Review. 2005;24:33-60.

[35] N Jansen, B Könighofer, S Junges, R Bloem, Serban A. Safe Reinforcement Learning Using Probabilistic Shields. 31st International Conference on Concurrency Theory. CONCUR. 2020;171:3:1–3:16.

[36] L. P. Kaelbling, M. L. Littman, A. R. Cassandra. Planning and Acting in Partially Observable Stochastic Domains. Artificial Intelligence. 1998;101:99–134.

[37] V. Mnih, K Kavukcuoglu, D Silver, A A Rusu, J Veness. Human-Level Control Through Deep Reinforcement Learning. Nature. 2015;518:529–533.

[38] J Schulman, F Wolski, P Dhariwal, A Radford, O Klimov. Proximal Policy Optimization Algorithms. Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017:1-12.

[39] Y. Sun, Z. Han, W. Yu, K. J. R. Liu. Trust management in wireless sensor networks: A survey and taxonomy. IEEE Communications Surveys / Tutorials. 2016;16:2508–2528.

[40] D Maoujoud, G Rens. Reputation-Driven Decision-Making in Networks of Stochastic Agents. Proceedings of BNAIC/BeneLearn. 2020. arXiv preprint: https://arxiv.org/pdf/2008.11791v1.

[41] M Alshiekh, R Bloem, R Ehlers, B Konighofer, S Niekum, et al. Safe Reinforcement Learning via Shielding. Proceedings of the AAAI Conference on Artificial Intelligence. 2018;32:2669-2678.

[42] D Bethell, S Gerasimou, R Calinescu, C Imrie. Safe Reinforcement Learning in Black-Box Environments via Adaptive Shielding. Proceedings of the 2025 International Conference on Learning Representations. ICLR. ResearchGate. 2025.

[43] W C Yang, G Marra, G Rens, L De Raedt. Safe Reinforcement Learning via Probabilistic Logic Shields. Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence. IJCAI-23. 2023;637:5739–5749.

[44] A Doucet, N de Freitas, N Gordon. Sequential Monte Carlo Methods in Practice. 1st edition. Springer Nature. 2001.

[45] I Sharafaldin, A H Lashkari, A A Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. International Conference on Information Systems Security and Privacy. ICISSP. 2018;1:108–116.

[46] A S Tanenbaum, D J Wetherall. Computer Networks. 5th ed. Pearson .2011.

[47] C Wang, C Redino, R Clark, A Rahman, S Aguinaga, et al. Leveraging Reinforcement Learning in Red Teaming for Advanced Ransomware Attack Simulations. In 2024 IEEE International Conference on Cyber Security and Resilience. CSR. 2024:262-269.

[48] A Agarwal, Y Song, W Sun, K Wang, M Wang, et al. Provable Benefits of Representational Transfer in Reinforcement Learning. The Thirty Sixth Annual Conference on Learning Theory. 2023: 2114-2187.

[49] Y L Khaleel, M A Habeeb, A S Albahri, T Al-Quraishi, O S Albahri, et al. Network and Cybersecurity Applications of Defense in Adversarial Attacks: A State-Of-The-Art Using Machine Learning and Deep Learning Methods. Journal of Intelligent Systems. ResearchGate. 2024;33:20240153.

[50] C M Bishop. Pattern Recognition and Machine Learning. Springer. 2006.

[51] J Platt. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Like-Lihood Methods. Advances in Large Margin Classifiers. 1999;10:61–74.

[52] T. Chen, C. Guestrin. XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016:785–794.

[53] I Goodfellow, Y Bengio, A Courville, Deep Learning. MIT Press. 2016.

[54] https://www.semanticscholar.org/paper/Discriminative-Particle-Filter-Reinforcement-for-Ma-Karkus/dc0276b62e8e92dc475bae5e107524749b948873

[55] A. Sinha, A. Mahajan. Agent-State Based Policies in POMDPs: Beyond Belief-State MDPs. 2024 IEEE 63rd Conference on Decision and Control. Tutorial Paper. 2024:6722-6735.

[56] A Raffin, A Hill, A Gleave, A Kanervisto, M Ernestus, et al. Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine Learning Research. 2021;22:1–8.

[57] https://lucykuncheva.co.uk/Combining$_P attern_C lassifiers_M ethods_a nd_A lgorithms_2 nd_e d_K uncheva$

[58] https://www.geeksforgeeks.org/machine-learning/a-comprehensive-guide-to-ensemble-learning/.

[59] https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[60] K Begović, A Al-Ali, Q M Malluhi. Cryptographic Ransomware Encryption Detection: Survey. Computers Security. 2023;132:103349.

[61] https://api.pageplace.de/preview/DT0400.9781351414388$_A$37405910/$preview$ – 9781351414388$_A$37405910.pdf.

[62] S Kullback, R A Leibler. On Information and Sufficiency. The Annals of Mathematical Statistics, 1951;22:79–86.

[63] L Chen, K Lu, A Rajeswaran, K Lee, A Grover, et al. Decision Transformer: Reinforcement Learning via Sequence Modeling. 2021. arXiv preprint: https://arxiv.org/pdf/2106.01345v1.

[64] J García, F Fernández. A Comprehensive Survey on Safe Reinforcement Learning.Journal of Machine Learning Research. 2015;16:1437–1480.

[65] Y Wang, M Xu, L Shi, Y Chi. A Trajectory Is Worth Three Sentences: Multimodal Transformer for Offline Reinforcement Learning. Proceedings of the Thirty-ninth Conference on Uncertainty in Artificial Intelligence. UAI. 2023;216:2226–2236.

[66] Z Zhuang, D Peng, J Liu, Z Zhang, D Wang. Reinformer: MaxReturn Sequence Modeling for Offline RL. 2024. arXiv preprint: https://arxiv.org/pdf/2405.08740v1.

Appendix A. RL Algorithms Configurations

PPO Configuration. PPO uses experience replay (buffer size 10^5), a clipped surrogate objective with clip parameter $\epsilon = 0.2$, learning rate 3×10^{-4}, GAE parameter $\lambda = 0.95$, and update epochs of 10 per rollout batch. The policy network consists of two hidden layers [256, 128] with tanh activations. The Red agent, when learning-based, uses the same PPO configuration to ensure fair comparison. PPO agents are trained in an offline manner at the end of each simulation episode.

DQN Configuration. DQN uses experience replay (buffer size 10^5), target network updates every 1000 steps, ϵ-greedy exploration with ϵ annealing from 1.0 to 0.01 over 50,000 steps, learning rate 10^{-4}, and discount factor $\gamma = 0.99$. The Q-network uses two hidden layers [256, 128] with ReLU activations. DQN agents are trained in an offline manner at the end of each simulation episode.

A2C Configuration. A2C uses n-step returns with $n = 5$, learning rate 7×10^{-4}, entropy coefficient $\beta = 0.01$ for exploration, and shared network architecture [256, 128] for both actor and critic. A2C agents are trained in an online manner at the end of each simulation step.

Appendix B. 14-node Environment

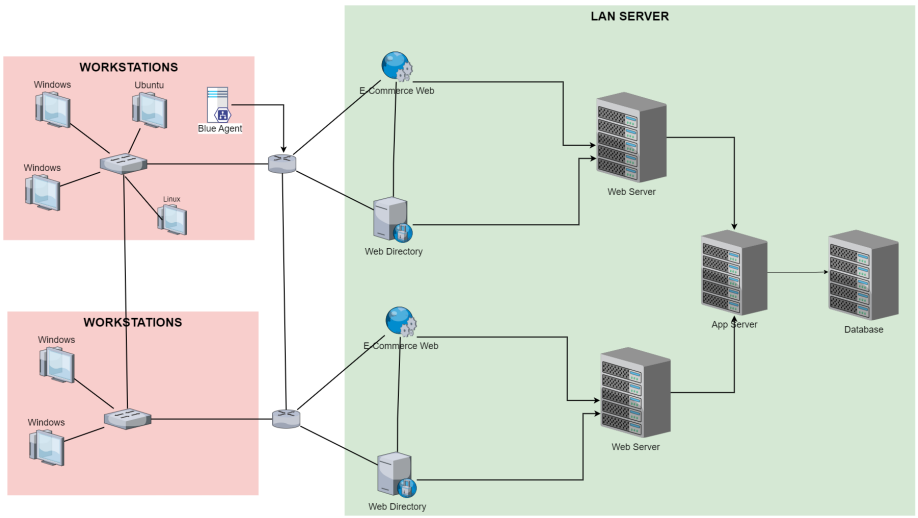

Figure 12: 14-node LAN