Leveraging Deep Reinforcement Learning for Cyber-Attack Paths Prediction: Formulation, Generalization, and Evaluation

Franco Terranova
franco.terranova@inria.fr
Université de Lorraine, CNRS, Inria, LORIA
Nancy, France

Abdelkader Lahmadi
abdelkader.lahmadi@loria.fr
Université de Lorraine, CNRS, Inria, LORIA
Nancy, France

Isabelle Chrisment
isabelle.chrisment@loria.fr
Université de Lorraine, CNRS, Inria, LORIA
Nancy, France

ABSTRACT

Attack paths represent the sequences of network nodes compromised by attackers while exploiting their respective vulnerabilities. Current methods for predicting such attack paths largely depend on existing human expertise or established heuristics. These traditional methods are time-consuming and require highly skilled threat-hunting analysts to identify these attack paths and proactively apply security measures. However, the task becomes challenging when facing large-scale and highly vulnerable networks. In this paper, we propose an alternative approach leveraging Deep Reinforcement Learning (DRL) techniques aiming to approximate the decision-making of attackers. Our approach embodies the attacker's perspective and tactics to leverage discovered paths for proactive security analysis and establish defense strategies. We introduce a novel re-formulation of the problem with a local view for the DRL agent, representing the source and target node of the attack at each timestep. Additionally, our training methodology involves a diverse set of network topologies of different sizes and exploitable vulnerabilities, demonstrating the ability of DRL algorithms to navigate topologies, identify attack paths, and compromise nodes. Results highlight the capability of the learned policies to generalize within entirely new topologies, arriving to discover 80% ± 0.08% of the attack paths in 1500 steps.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; **Network security**; • **Computing methodologies → Planning under uncertainty**; **Sequential decision making**.

KEYWORDS

Cyber-Attack Paths Prediction, Vulnerability Assessment, Deep Reinforcement Learning, Proactive Security, Automated Penetration Testing, Deep Learning, Machine Learning, Artificial Intelligence

ACM Reference Format:
Franco Terranova, Abdelkader Lahmadi, and Isabelle Chrisment. 2024. Leveraging Deep Reinforcement Learning for Cyber-Attack Paths Prediction: Formulation, Generalization, and Evaluation. In *The 27th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2024), September 30–October 02, 2024, Padua, Italy.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3678890.3678902

1 INTRODUCTION

Intrusion and attack detection are among the oldest and most common problems in cyber security tasks [27]. However, these approaches are only reactive to observed patterns or learned detection models. Proactive and predictive methods can be more valuable in reacting to attacks before they happen. In particular, both detection systems and proactive security measures gain efficiency when knowing the attack paths that may be used by attackers to gain access to a network. By identifying the sequences of vulnerabilities and attack techniques that attackers exploit to achieve their objectives, these systems can enhance their effectiveness.

A relevant challenge in attack path prediction [27] is its dependency on the expertise of professionals skilled in penetration testing (PT), security analysis, and threat hunting. This reliance on human expertise not only constrains the assessment's efficiency but also its scalability when facing large-scale and highly vulnerable networks. In such situations, the security analysts are overwhelmed and the task of prioritizing the vulnerabilities to fix becomes challenging. Attack prediction tools may provide network security teams insights about potential future attacks or malicious activities, to prepare defense strategies [14]. Specifically, the process of forecasting an attacker's steps during attack propagation or multi-stage attacks is also known as *intrusion prediction* [1].

Recent advances are studying how Machine Learning (ML) methods can approximate the strategic behavior of an attacker, trying to encapsulate the decision-making process of this behavior within the parameters of an ML-based model. Among the different categories of methods, time-series analysis models, Markov models [13], Bayesian networks [45], and Deep Learning (DL) models [16] are showcasing great potential in predicting future cyber-attacks [27].

With these approaches, studies want to prove the potential of a function approximator to execute the appropriate exploits and move through the network, aiming at discovering and/or taking control over its devices. The resulting model may enhance cyber-situational awareness [27], crucial for both assessing the current resilience of the network, but also to determining how to strengthen network security by identifying the most common or critical nodes and paths. However, many of these existing techniques still have scalability and generalization issues, in particular for predicting attack paths in large-scale and highly vulnerable networks.

RAID 2024, September 30–October 02, 2024, Padua, Italy

ACM ISBN 979-8-4007-0959-3/24/09
https://doi.org/10.1145/3678890.3678902

Recently, Reinforcement Learning (RL) [50], an ML paradigm that appears to be suited for modeling the sequential nature of cyber-attack discovery, has been used for attack path prediction with automated penetration testing or exploring attack graphs to find paths that may be exploited by attackers. However, there is limited knowledge regarding the performance of the available RL techniques in such tasks, how their models can be generalized, and how they can be evaluated comprehensively and reliably.

In this paper, we leverage RL techniques and specifically their DL variants to train an agent for learning a policy able to identify attack paths in unseen vulnerable networks during its training phase. The RL agent maps states to actions, to maximize cumulative rewards over time while using a Markov Decision Process (MDP) [44]. A MDP provides a formal framework to capture the sequential decision-making, characterized by a series of states (representing the topology's configurations at given time points), actions (possible interventions or movements that the attacker might execute), and rewards (outcomes of actions aimed at evaluating their effectiveness). By integrating DL [17] techniques, we leverage neural networks (NNs) [20] as function approximators capable of discerning complex patterns representations in the state and action spaces, shifting to the paradigm of deep RL (DRL), and potentially getting closer to approximate complex decision strategies, typical of skilled attackers. In the RL framework, the agent must interact with its environment to learn, in a manner that conforms to the structure of a MDP.

One possible approach to automate the identification of attack paths involves creating an attack tree model from the topology and the identified vulnerabilities, and then using RL to identify the most effective branches for attack [26]. However, this method heavily depends on the presence of a graph that is either manually created by humans or generated by another software, such as MulVAL [40]. Additionally, it assumes complete knowledge of the network's overall topology, which may be infeasible in practice for large and complex networks. Thus, we followed in this work the direction of studies based on recent tools [51] [31] [5] [23] that target the full automation of the vulnerability assessment process, eliminating the dependency on pre-existing attack graphs or comprehensive knowledge of the network's topology. This shift in methodology transforms the classic MDP into a Partially Observable MDP (POMDP) [29]. With this modification, the simulation will provide partial observations that reflect the current state of the network discovery process. In real-world scenarios, attackers also possess a limited and evolving understanding of the topology, gradually expanding their knowledge during the process.

1.1 Key Contributions

This paper aims to reformulate and refine the application of DRL for predicting cyber-attack paths, with a particular focus on improving the model's generalization capabilities across different network topologies. This enhancement brings the model closer to practical deployment. The following points provide a detailed summary of the key contributions of this paper:

- **A topology independent approach with an attacker's local view**: Our approach involves a novel reformulation of the conventional choices for state and action spaces used in previous research studies for this task [4] [19] [55] [42]. This enables the training and evaluation of a single NN-based cyber-attacker agent across diverse topologies of varying sizes. To achieve this, we propose a local view of the agent's state space, using a source and target node instead of a global view of the overall topology. This modification also simplifies the action space by requiring the agent to choose only the vulnerability to exploit between the two nodes or a movement action for the two anchors, rather than selecting the nodes of interest as well. Therefore, there is no need to adjust the number of neurons in the NN input and output layers as the size of the graph changes. This offers a more flexible DRL paradigm that enables training and evaluation without being limited to a specific fixed topology size. As a result of this advancement, there is no longer a need to retrain a new agent for each topology scenario.
- **A more generalizable training & evaluation approach**: We propose a more versatile training and evaluation approach using a periodic switching module, allowing the agent to be exposed to multiple topologies representing various scenarios with different topology sizes, vulnerabilities, firewall conditions, and starting nodes. These scenarios' parameters are selected from predefined probability distributions, ensuring systematic variation in training and evaluation conditions. By doing so, our approach aims to significantly improve the agent's ability to generalize across diverse environments, thereby trying to bridge the gap between simulated scenarios and real-world complexities. Unlike traditional approaches that have trained agents on a single static topology with a fixed starting node, this approach aims to avoid bias and enables robust evaluation of the agent's strategy.
- **Performance evaluation across episodes**: By periodically varying both the topology and the starting node, we influence the agent's maximum achievable performance in each episode and its subsequent evaluation. To handle this variability, we propose new graph abstractions that assess the complexity of each topology for a DRL agent, while also determining the same complexity for a specific starting node. This approach allows us to compute the maximum achievable score for the agent under each starting condition and ensure a proper result integration across multiple starting conditions during evaluation.
- **Tool modifications and enhanced evaluation framework**: We are releasing an enhanced version of Microsoft CyberBattleSim [51], a tool that provides structured environments for training and evaluating RL-driven attackers. The tool modifications alongside the network topologies utilized in the study, are comprehensively linked within the paper [52] [53]. This update incorporates the enhancements mentioned in the previous points and introduces an interface for utilizing Stable-Baselines3 (SB3) [46] algorithms and RLiable [2] indicators. Combining the previous modifications with the integration of these well-established RL libraries enables a thorough evaluation of DRL algorithms in the cyber-attack path prediction task. Additionally, we introduce domain-specific metrics related to individual actions and identified attack paths to further enrich the evaluation framework.

The rest of this paper is organized as follows. Section 2 provides an overview of RL and the algorithms employed in our study. Section 3 reviews existing literature on the application of RL to automated cyber-attack prediction. In Section 4, we outline our proposed methodology. Section 5 presents a comprehensive experimental setup, while Section 6 conducts a benchmarking analysis to evaluate and compare algorithms' performances. Section 7 discusses the potential deployment pipeline and its limitations for network analysts, while Section 8 concludes with the implications of our findings and suggests directions for future research.

2 REINFORCEMENT LEARNING

In this section, we provide background information on RL and the characteristics of the algorithms covered in the study. RL, alongside supervised and unsupervised learning, is one of the paradigms in the general context of ML.

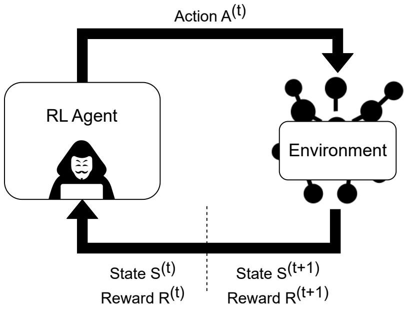


Figure 1: Overview of the main RL elements: agent, environment, states, actions, and rewards.

The main components of the RL paradigm (Figure 1) include the agent, environment, rewards, actions, and states. At each time step t, the agent exists in a state $S^{(t)}$, takes an action $A^{(t)}$, and receives from the environment a transition to a new state $S^{(t+1)}$ and the reward $R^{(t+1)}$ associated with the action taken in the previous state. Learning occurs via a process of trial and error, using the reward feedback from the environment in response to the actions it executes across various input states. This loop empowers the agent to discover optimal behavior within the environment by interacting with it. In the RL realm, an *episode* represents a "game trial" of the learning agent interacting with the environment. It starts with the agent in an initial state, involves a series of actions taken by the agent, transitions between states, and concludes when a termination condition is met (e.g. all reachable nodes have been owned) or a cut-off of steps is reached. A policy is defined to be optimal if it selects actions that maximize the cumulative expected rewards over the entire time horizon.

2.1 Markov Decision Process

The environment is typically represented as an MDP, an extension of a Markov Process (MP) [39]. The MDP formulation is given by the components highlighted in Tuple 1.

$$\begin{aligned}
\text{MDP:}\quad & (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma) && (1)\\
\mathcal{S}:\quad & \text{State space} && (2)\\
\mathcal{A}:\quad & \text{Action space} && (3)\\
\mathcal{P}:\quad & \mathcal{S} \times \mathcal{A} \to \mathcal{S}, \text{ State transition function} && (4)\\
\mathcal{R}:\quad & \mathcal{S} \times \mathcal{A} \to \mathbf{R}, \text{ Reward function} && (5)\\
\gamma:\quad & \text{Discount factor} && (6)
\end{aligned}$$

Additionally, a trajectory (τ) is defined as a sequence of states, actions, and rewards that an agent experiences as it interacts with an environment. Specifically, a trajectory can be represented as a sequence $(s_0, a_0, r_1, s_1, a_1, \ldots)$. As previously mentioned, the objective of the agent is to maximize the expected cumulative reward, known also as the return, for the trajectory. The return from a state at time t over a trajectory τ (assuming an episode of finite length N) is defined as delineated in Equation 7. This calculation aggregates the rewards accumulated at every timestep throughout the trajectory.

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{N} \gamma^k r_{t+k+1} \tag{7}$$

Equation 7 also underscores the significance of the discount factor γ (Equation 6), a scalar value that determines the importance of future rewards compared to immediate rewards in the optimization process. It plays a crucial role in the agent's decision-making by influencing how much weight is given to future rewards.

2.2 Partially Observable Markov Decision Process

A POMDP extends the MDP framework to scenarios where agents have limited information about the environment's state. Defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Omega, O, \gamma)$, a POMDP incorporates in addition:

- Ω, a finite set of observations that the agent can perceive, providing partial information about the current state of the environment.
- O, the observation probability function, which specifies the probability of observing a particular observation given an action and the resulting state.

Unlike MDPs, in POMDPs, agents receive observations rather than full-state information, necessitating learning agents that can handle uncertainty and partial knowledge to make decisions.

2.3 Model-based vs Model-free Methods

Model-based methods [38] [50] aim to directly approximate the state transition function, denoted as $\mathcal{P}$ (Equation 4), and the reward function, $\mathcal{R}$ (Equation 5). Given the complexity of accurately predicting these functions, which knowledge leads to the solution of the MDP, direct approximation of these functions for real-world tasks is challenging. Consequently, this complexity often leads to the utilization of model-free methods [50] in real-world tasks, which overcomes the need to directly approximate $\mathcal{P}$ and $\mathcal{R}$ by focusing on other strategies that learn policies or value functions. Model-free methods include several classes of algorithms:

- *Value-based methods*, such as Deep Q-Networks (DQNs) [37], focus on learning a value function. Typically, the Q function is learned, $Q(s, a)$ (Equation 8), which quantifies the value of taking an action a in a given state s. This function enables the selection of actions that maximize the return.

$$Q(s, a) = \mathbb{E}_{\tau\sim(s,a)} \left[R(\tau) \mid s_0 = s, a_0 = a\right] \quad (8)$$

- *Policy-based methods*, including Proximal Policy Optimization (PPO) [49] and Trust-Region Policy Optimization (TRPO) [48], directly learn a policy function π that approximates a probability distribution on the action space for each state (Equation 9), still aiming for a policy that maximizes the return.

$$\pi(a|s) = P(a_t = a|s_t = s) \quad (9)$$

- *Actor-critic methods*, like the Advantage Actor-Critic (A2C) [36], combine both value-based and policy-based approaches by maintaining two models: one for the policy (the actor) and another for the value function (the critic).

Given the complexity involved in predicting cyber-attacks, we will directly utilize model-free methods for this task, in line with the approach taken by all existing studies on the topic.

3 RELATED WORK

Current techniques for forecasting cyber-attacks fall into various categories, including discrete, continuous, ML-based, and other strategies [27]. In the group of discrete techniques, graph models are predominant, especially ones leveraging attack graphs, Bayesian networks, or Markov models.

RL-based approaches lie at the intersection between Markov models and the larger category of ML-based approaches. More recently, these RL-based techniques have been adopted in several works for cyber-attack prediction, in particular for autonomous PT to discover how attackers could make intrusions on a given computer network. Initial studies on autonomous PT [24][54] started with the assumption of the availability of the topology of the network or an attack tree, with solutions for which input data may be impractical to acquire in real-world scenarios for large-scale complex networks. More recent works are going towards a black-box approach, focusing on dynamically discoverable environments, allowed by simulation platforms like CyberBattleSim. Given the challenges in accessing data on network vulnerabilities due to security and privacy concerns, there is a growing need for these simulation tools capable of generating network topologies embedded with vulnerabilities [7]. Additional environments that merit mention include CybORG [5], CyGIL [31], and AutoPentest-DRL [23]. Each of these platforms offers an interface compatible with OpenAI Gym [9], facilitating the training of RL agents. In this work, CyberBattleSim is employed as the simulation tool of choice, due to its support to the realistic evolving representation of the attack process and its support for various scalable communication topologies.

In the recent research outcomes, a notable gap exists regarding the utilization of state and action space representations specifically aimed to be topology independent, as well as the training/evaluation on different topologies in a scalable fashion. The two problems are strictly connected, since a topology-dependent NN cannot be applied on different topologies, requiring to retrain a different model for each topology. A commonly adopted approach for structuring the input layer of the NN involves translating the network topology into an array, which length is proportional to the quantity of nodes. These nodes are sequenced based on the order in which they are discovered. This approach results in a topology-dependent input layer's size, where also the ordering of the input changes based on the unpredictable sequence of node discovery, deteriorating learning valuable mappings from the input data. Similar implementations and similar challenges are present in the output layer, which typically includes the selection of nodes and the attack. This approach also presents practical challenges as the topology size increases, since the input and output space will vary. It leads to scalability issues and instability in research findings, as conclusions may vary largely with changes in the input and output layers' dimensions. Significant challenges are present as well in using NNs to transfer learned behaviors across various topologies, due to the lack of consistent semantics in the ordering of nodes. Furthermore, this approach does not account for dynamic changes in topology size, a characteristic common in highly dynamic environments like the Internet of Things (IoT), becoming a critical limitation in applying these NN-based models to real-world environments. The current solutions in the literature and the remaining existing drawbacks have been described in more detail in Table 1. Finding a solution to these problems may facilitate model migration, generalization, and consequent deployment.

Solutions like masking actions and padding to standardize input/output sizes among topologies have been proposed [19] [32], yet they fail to address the issue of inconsistent node ordering semantics among topologies. Moreover, present studies primarily focus on a training process applied within a singular network topology and from the same starting node compromised by an attacker. A limited number of papers have begun to examine the agent's transferability across diverse topologies [32] [4] [19]. However, there remains a significant gap in research dedicated to achieving this in a scalable manner, either across a large number of diverse topologies or through robust evaluation methods.

Our source-target approach for the attacker's local view coupled with a scalable training/evaluation strategy wants to cover the existing drawbacks by making the RL agent learning process more topology independent and trying to approximate general knowledge in the agent.

Moreover, different sources of non-determinism, such as extrinsic factors (e.g. hyper-parameters) and intrinsic factors (e.g. random seeds and the number of runs), may strongly affect RL results [10] [21]. Relying on less statistically robust metrics can lead to considerable variations in research findings, hence we leverage more reliable alternatives to mean and standard deviation for more accurate assessments [2].

Furthermore, it has been noted that variations in the implementation of identical algorithms can lead to vastly different outcomes [10]. To reduce this variability and improve the reproducibility of findings, this study has chosen to use reliable implementations from the SB3 library [46]. Moreover, existing research might exhibit statistical biases due to the insufficient exploration of hyper-parameters in DRL algorithms. Recognizing this gap, our study has

Table 1: Summary of challenges and approaches in recent papers and implementations regarding the application of RL for attack prediction.

Approach	Using the Approach	Still Present Drawback
Topology-dependent state and/or action space	[4], [19], [55], [42]	Challenges in scalability (for large topologies), evaluation (different results varying input and output layers' sizes), and transferability on different topologies.
Encoding in state and action spaces dependent on node discovery order	[4], [19], [55], [42]	Difficulty in learning order-invariant patterns across topologies or across several episodes where the order of discovery is different.
Masking invalid actions in the action space	[19], [32]	May deteriorate learning since the invalidity of actions is dependent on the order of nodes' discovery.
Padding techniques to fix a maximum number of nodes in state and action space	[19], [32]	Trade-off between upper limit's size and the inefficiency of unused capacity.
Single node encoding and global features as state	Agent Wrapper Implementation of CyberBattleSim tool, [33]	Agent with limited view on the target of the attack, potentially not being able to learn a full mapping.
Transfer learning on new topologies	[32] [4] [19]	Gap in scalable evaluation across diverse topologies or robust evaluation methods.

incorporated an initial phase of hyper-parameters optimization, before proceeding to the comparative analysis of the algorithms.

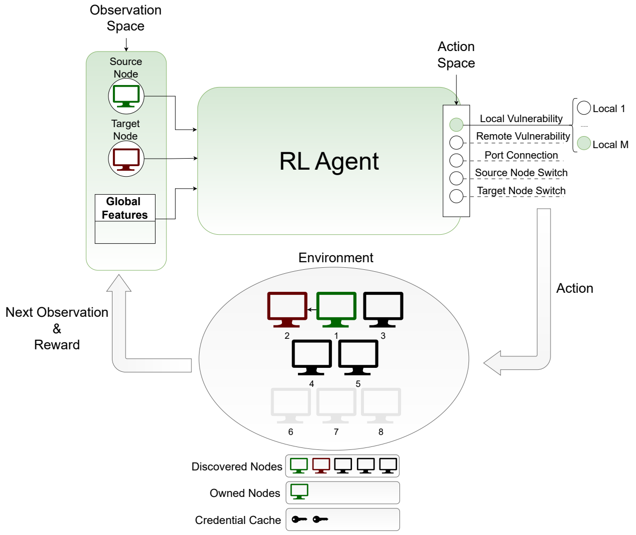


Figure 2: A more detailed view of the RL process: the agent selects the optimal action based on the observation.

4 METHODOLOGY

In this section, we discuss our topology independent and scalable MDP formulation, as well as the process of generation of the environment used, along with the training and evaluation pipeline followed.

4.1 Markov Decision Process with a Local Perspective

The MDP framework implemented in the work adopts a source-target node strategy, enabling transferability across varying network topologies and independence on the node ordering in the topology and across topologies. The progression of the episode is managed through three primary lists: the *owned* list, the *discovered* list, and the *credential cache*. The *owned* list contains nodes that the agent has successfully compromised throughout the episode. The *discovered* list includes nodes that have been identified, typically through exposure of their node IDs via vulnerabilities. Lastly, the *credential cache* stores credentials obtained by exploiting vulnerabilities, and they are assumed to be *(nodeID, port, password)* tuples. At the beginning of each episode, the *owned* and *discovered* lists include only the starter node, while the credential cache starts empty. The game will allow interactions between the source node (initially the starter node) and the target node (initially replaced by a dummy blank node). As the game progresses, the agent's actions (initially only with local actions as valid) lead to the discovery of nodes, credentials, and the compromise of nodes, dynamically updating the lists and the potential choices for the source and target nodes.

4.1.1 Action Space. The action space, defined within the MDP's local perspective, includes:

- **Local Vulnerabilities:** An action corresponding to each local vulnerability, targeting the source node.
- **Remote Vulnerabilities:** An action for every remote vulnerability, targeting the target node.
- **Port Connections:** Actions for establishing connections from the source to the target node, one per available port. We operate under the assumption that, upon the agent's selection of a port, it will subsequently attempt access using all matching credentials stored in the cache.
- **Navigational Actions:**
  - Source node switching, both forward and backward within the *owned* list.
  - Target node switching, forward and backward within the *discovered* list.

The agent will start to explore nodes in the topology via the source-target approach and take action on them, potentially enlarging the list of discovered nodes and owned nodes if the right actions have been taken. The simulation's environment dynamically updates the configuration of the game treating the lists as circular queues for switching actions.

4.1.2 Observation Space. The observation of an agent at any given time step is delineated by the following components:

- **Source Node Representation:** Attributes of the source node.
- **Target Node Representation:** Attributes of the target node.
- **Global Features Array:** A set of summarized metrics reflecting the previous lists, comprising:
  - *Number of Discovered Nodes:* Total nodes identified by the agent.
  - *Number of Owned Nodes:* Nodes that have been compromised by the agent.
  - *Number of Discovered Credentials:* The count of unique credentials discovered, indicating the size of the credential cache.
  - *Average Node Value:* The average value across all discovered nodes. It may suggest the agent switching the target node if it may be less valuable than the others available.
  - *Exploitable Vulnerabilities:* The number of yet-to-be-exploited vulnerabilities within the owned nodes. It may suggest focusing more on already compromised nodes rather than exploring new nodes.
  - *Accessible Ports:* Count of ports known to be accessible in discovered nodes (information obtained from the credential cache via lookup). It may suggest focusing more on port connections to already discovered nodes.

A compact representation of the MDP is presented in Figure 2. Details on the node representation will be provided in Section 4.2.

4.1.3 Reward function. While learning, the RL agent should learn to optimize several objectives, such as the number of nodes that have been discovered or compromised, the credentials gathered, or the number of actions taken. The reward function R used in this study is based on CyberBattleSim and represents the weighted linear combination of the outcomes of a certain action a in a given state s, showcasing the control gained by the agent as a consequence of the exploitation of a vulnerability (Equation 10). The weights within the reward function can be adjusted to emphasize different objectives, such as discovery, control, or node-specific targets.

$$\begin{aligned} R(s,a) = & \left(\sum_{i \in \text{nodes owned(s,a)}} K_{\text{value}} \cdot \text{value}(i) \right) \\ & + K_{\text{node discover}} \cdot |\textit{nodes discovered(s,a)}| \\ & + K_{\text{credential discover}} \cdot |\textit{credentials discovered(s,a)}| \\ & - K_{\text{cost}} \cdot cost(a) \\ & + K_{\text{first success}} \cdot \textit{first success(s,a)} \\ & - penalty(a) \end{aligned} \tag{10}$$

The components of the reward function take into account all the possible outcomes of the agent's interaction with the current source-target node pair. The first term accounts for the sum of values of the nodes owned as a result of the action taken. The other terms encourage nodes' discovery and credentials gathering, penalize the costs associated with exploiting vulnerabilities, and reward first-time successful actions. The vulnerability cost could be quantified in terms of time needed, computational effort, or increased risk of detection. It ensures that the agent considers the efficiency and risks associated with different actions. This term also allows for the adjustment of cost weights for local and remote vulnerabilities, as well as differentiation based on the protocol of each vulnerability. The penalty term is determined based on whether the action taken adheres to a predefined set of patterns, such as suspicious activity, failed exploits, or repeated actions.

4.1.4 Threat Model. In our methodology, our primary goal is to identify goal-agnostic attack paths. To achieve this objective, we have tailored the reward function coefficients (detailed in Section 4.1.3) to guide the agent toward maximizing general control over nodes throughout the topology, without targeting specific nodes (further discussion on the specific coefficient values used is present in Appendix A.2). In fact, if the agent is intended to gather general defense insights without prior knowledge of specific attacker goals — knowledge that is often unrealistic a priori — training it for goal-agnostic paths is preferable [30].

4.1.5 Normalization strategy. To enhance the stability and effectiveness of the training process, both the observation vector and reward signal received at each iteration by the agent are normalized using *z-score normalization*, which has been well demonstrated to facilitate more consistent training outcomes and improved algorithm performance [25]. This approach is based on the *VecNormalize* technique from the SB3 library.

4.2 Node Representation

In this subsection, we expand upon the previously outlined local MDP framework utilized in this work, specifically focusing on the observation representation which includes encodings for both source and target nodes. Table 2 details the attributes that define the node representation of our methodology. These attributes will be encoded to fully represent a specific node in the topology and will work as the array representation of a node for the agent. The partial observability characteristic emphasizes attributes that modify the agent's visibility into each node, changing depending on whether the node has been compromised, introducing an extra layer of partial observability within an inherently partially observable environment. This adaptation has been done on the assumption that, in a practical deployment scenario, some attributes might be obtained through scanning tools like Nmap [34]. However, detecting certain vulnerabilities could necessitate higher access permissions to the host, becoming visible only after the node is fully compromised. We assume that details such as the node's value and the availability weights of services, which may not be directly derivable through existing tools in a potential deployment phase, could be approximated by the agent based on contextual factors. In this study, they are replaced by respectively a random value in the range [0, 100] and default importance weights set by the simulation tool.

4.2.1 Approaching the Markovian Ideal. To ensure the environment's Markovian property, stating that future states depend solely on the current state, additional properties have been incorporated (Table 2). These properties help differentiate between scenarios

Table 2: Encoding attributes of the communication networks' nodes and their properties.

Attribute	Description	Partial Observability	Additional Markovian Property
Initial Node	Identifies if this node was the starting point for the current episode.	X	X
Node Value	A numerical value reflecting the node's intrinsic importance in the topology.	X	X
Listening Services Array	Array with elements identified by a port identifier, each including status and weight of importance.	X	Each entry with a property indicating if the port is accessible with the current credentials.
Vulnerability Array	Array with elements characterized by type (local/remote), outcome (nodes discovery, etc.), and exploitation cost.	Each entry is hidden until a correct exploitation attempt is made or the node is fully compromised.	Each entry with a property indicating if it was already exploited or tried (to avoid the agent repeating redundant actions).
Node Status	Indicates the node's current operational state (running, stopped).	X	X
Privilege Level	Denotes whether the node has been compromised.	X	X
Service Availability	Represents the node's importance in network service levels.	X	X
Firewall Settings	Rules specifying ALLOW or BLOCK actions for each protocol and direction.	X	X

where the observation representation might appear identical yet lead to divergent outcomes due to unseen information. This assumption is crucial for the convergence of most of the well-known RL algorithms. Indeed, as it is going to be detailed by the vulnerability assignment in Section 4.3.2, the game inherently lacks Markovian properties. For example, a granted connection to a specific port depends on previous actions, such as exploiting a vulnerability on an adjacent node to obtain valid credentials for that port. Including the extra Markovian property can enhance learning stability by incorporating information that may be invisible otherwise.

4.3 Topologies Generation

This section will delineate the methodologies employed to generate the network topologies used for training, a clearer picture of the resulting MDP framework, and additional abstractions added to the foundational model to discuss the complexity of a topology for the RL agent and to determine the potential impact of the agent on the topology from each starter node.

4.3.1 Communication graph. The environment constructs a network communication graph using a set of protocols available for communication among nodes. For every protocol, there exists a designated block of clients and servers of customizable sizes. The connections between nodes belonging to the two groups are determined by probability distributions. Two types of Beta distributions [28] are used to model the likelihood of network connections: an *intra-beta distribution* and an *inter-beta distribution*. Characterized by two shape parameters, α and β, the Beta distribution's shape varies depending on these parameters. It exhibits a positively skewed distribution when $\alpha < \beta$ and a negatively skewed distribution when $\alpha > \beta$.

- The *inter-beta distribution* determines the probability of connections between different groups, such as an RDP client communicating with an RDP server.
- The *intra-beta distribution* estimates the probability of connections within the same group, for instance, that an RDP client establishes connections with another RDP client, hence also serving as an RDP server. This probability distribution ensures an overlapping among the two groups.

4.3.2 Vulnerability assignment. Starting from the communication graph containing protocol edges among nodes, vulnerabilities are assigned among every pair of nodes according to a set of probabilities reflecting the potential vulnerabilities inherent to each protocol. The assigned vulnerabilities could result in the exposure of neighbors' node IDs, thereby unveiling their connected nodes, or the disclosure of neighbors' credentials, thus enabling the agent to access the neighbor nodes on the specific protocol. The specific vulnerability probabilities modeled in our study will be discussed in Section 5.1 and Appendix A.1. Additional probabilities will model realistic real-world scenarios:

- Probability of password change, that can invalidate credentials leaked through earlier probabilities.
- Probability of password reuse, that can enable the agent to apply the same credentials across different nodes and protocols.
- Probability of incoming and outgoing firewall BLOCK rules, designed to test the agent's resilience and adaptability in navigating and overcoming network restrictions.

In an ideal scenario, the agent is expected to learn to adopt a strategic pattern involving the exploitation of local or remote vulnerabilities generated through these probabilities. This strategy will allow credential gathering and node discovery. This should be concurrent to a dynamic interchange of source and target nodes until a port connection can be established with the current credentials in the cache and the firewall conditions, ultimately leading to the ownership of the target node. This newly conquered node then transforms into the agent's new source node.

4.3.3 Discovery and access graphs. The difficulty level of discovering and compromising a topology is conceptualized as a multi-parametric function that integrates various elements: the aforementioned probabilities, the count of nodes for each protocol, and the Beta distributions characterizing the probabilities of protocol presence. Determining the complexity of the topology based on all these parameters may be a difficult task. Obtaining this information is crucial, particularly before training an agent on the topology, as it provides insights into the maximum level of performance the agent might achieve in breaching the network topology. For this reason, two new graph abstractions and a graph metric are proposed in this paper to determine the complexity of a topology for the RL agent. The *discovery graph* (Equation 11) is defined as a graph with the same node set V of the communication graph, with edges between nodes u and v if there exists at least one vulnerability in u that reveals the node ID of v, thus making it discoverable.

$$\begin{aligned} G_{\text{discovery}} &= (V, E_{\text{discovery}}) \quad \text{where} \\ E_{\text{discovery}} &= \{(u,v) \in V \times V \mid \exists \\ &\quad \text{vulnerability in } u \text{ exposing } \mathrm{ID}(v)\} \end{aligned} \tag{11}$$

Analogously, the *access graph* (Equation 12) is defined as a graph with the same node set and with edges between nodes u and v if there exists at least one vulnerability in u that exposes a valid credential to v, thus making it accessible.

$$\begin{aligned} G_{\text{access}} &= (V, E_{\text{access}}) \quad \text{where} \\ E_{\text{access}} &= \{(u,v) \in V \times V \mid \exists \\ &\quad \text{vulnerability in } u \text{ exposing } \mathrm{Credential}(v)\} \end{aligned} \tag{12}$$

We also introduce a *connectivity* metric (Equation 13), denoted as C, which ranges between 0 and 1. This metric is designed to quantify the ease of navigating a graph from any given starting point.

$$C = 1 - \frac{\bar{L}}{2 \cdot (N-1)} \tag{13}$$

Here, $\bar{L}$ represents the average shortest path length across all node pairs, and N is the total number of nodes in the network. The shortest path between two non-reachable nodes is set to twice the maximum, $2 \cdot (N-1)$. C achieves a value close to 1 if, starting from any node within the graph, the shortest path required to reach any other node is 1, while the metric falls to 0 in scenarios where, regardless of the chosen starting node, no other nodes within the graph can be reached. Values of C between these two extremes reflect intermediate degrees of connectivity. To evaluate the overall complexity of a network's topology, we employ a weighted linear combination of the connectivity metrics derived from both the discovery and access graphs of the topology (Equation 14).

$$\textit{Topology Connectivity} = \alpha \cdot C(G_{\text{access}}) + \beta \cdot C(G_{\text{discovery}}) \tag{14}$$

Thus, if a vulnerability exists that allows direct discovery or access from the starting node to any other node within the graph, it results in a high value of C. Otherwise, if traversing the graph is necessary to reach other nodes, the connectivity score diminishes based on the extent of traversal required, approaching a value closer and closer to zero if many nodes cannot be discovered or accessed. Figure 3 summarizes the generation of a topology starting from the creation of the communication graph of a topology based on the protocols' parameters (Step 1), then used with the vulnerability probabilities to the generation of the discovery and access graphs (Step 2) of the same topology.

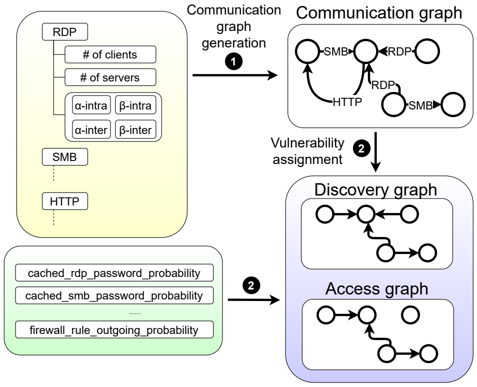


Figure 3: A summarized view of the generation of topologies: protocols' parameters generate the communication graph. The latter, paired with vulnerability probabilities, will determine the vulnerability assignment.

4.4 Training Pipeline

To enhance the agent's ability to generalize beyond the limitations of training on a single network topology and starting from a fixed node in each episode, this paper introduces a new training and evaluation pipeline designed to improve the agent's generalization capabilities, and redefine the evaluation strategy, as illustrated in Figure 4. The local MDP re-formulation indeed enables the same NN to be reused for training on new topologies. A wide array of network topologies has been created through the adjustment of hyper-parameters associated with the communication graph—such as the number of clients and servers, along with the parameters of Beta distributions for each protocol, as detailed in Section 4.3.1—and the vulnerability assignment, which includes varying the probabilities outlined in Section 4.3.2. Each hyper-parameter will vary in a certain range (more details in Appendix A.1), and each set of hyper-parameters values will produce a unique topology characterized by its specific level of connectivity C (Step 1). Creating diverse topologies with varying probabilities can bridge the gap between simulations and the real world, allowing agents trained on multiple configurations to develop more generalizable behaviors.

To explore the impact of topology connectivity on the performance of agents, we generated 500 network graphs, each containing between 50 and 100 nodes. The networks were categorized into five distinct groups based on their connectivity values, with each group consisting of 100 graphs (Step 2). The groups were organized according to connectivity ranges: [0, 0.2], [0.2, 0.4], [0.4, 0.6], [0.6, 0.8], and [0.8, 1.0]. The generation process was carefully controlled to populate each group with graphs whose connectivity scores

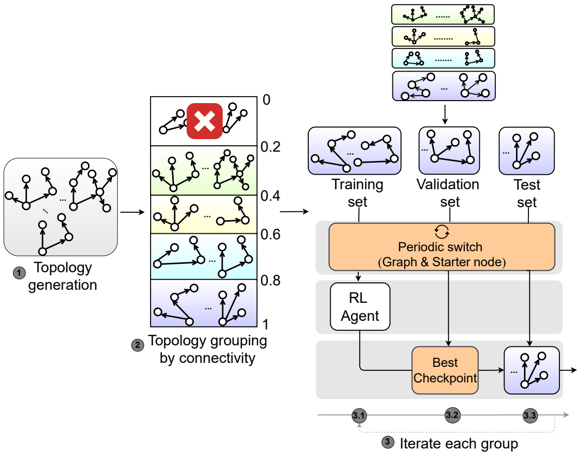


Figure 4: A summarized view of the training & evaluation pipeline: topologies are generated, segmented into groups, and each group undergoes training and evaluation.

were evenly distributed within the specified ranges. However, to maintain the integrity and interpretability of the results, we chose to exclude networks with extremely low connectivity (group [0, 0.2]) from our analysis. This decision was taken due to the absence of significant vulnerabilities within such topologies, which could skew the understanding of the agent's performance, leaving us with 4 distinct groups for the study. For each group, representing a specific level of difficulty that could be seen as a different level of a game, each RL algorithm has been subjected to 20 training runs. This choice was adopted based on insights from existing research in RL [10], suggesting that this number is the minimum need to achieve statistically reliable results when employing bootstrap confidence intervals (CIs) [12], the technique used by the indicators of the RLiable library.

At each run, the group of topologies is divided into a training set (60%), a set of topologies used to train the agent (Step 3.1), a validation set (20%), a set of topologies hidden and used periodically during training to evaluate generalization capabilities of the agent, and test set (20%), set of topologies used post-training to evaluate the capabilities of the agent. The validation set will be used to derive the best checkpoint of the agent's NN (Step 3.2), then evaluated on the test set, reserved a-posteriori as a different set to reduce the bias introduced (Step 3.3). The division of each group in these three sets will change every run.

Given a group with a connectivity range, the agent is run (trained or evaluated) across the multiple topologies simultaneously, alternating between them based on a fixed interval (denoted as *switch interval*) with the possibility of revisiting previously encountered topologies due to the replacement strategy. Switching with replacement ensures that a topology may be re-drawn later, a mitigation to the potential presence of catastrophic forgetting [15], typical of NNs when exposed to new experience.

Within each topology, kept for several episodes as defined by *switch interval*, the starting node is randomly changed with replacement every episode. Nodes that are unable to connect to a minimum threshold of the topology (set to 10%) are considered isolated and excluded from this random selection. The episode will conclude when a *cut-off* in terms of number of iterations is reached, or if the agent successfully reaches the termination condition: the agent captures all nodes accessible from its starting position. Achieving this goal triggers the agent with a bonus, a winning reward that will be added to the general reward, providing information about the conclusion with the success of the game.

5 EXPERIMENTAL SETUP

This section outlines the experimental setup that laid the groundwork for the results detailed in Section 6. It will cover the topologies generated by CyberBattleSim, describe the DRL algorithms and the NN used, and discuss the strategy for hyperparameter optimization that was employed to fine-tune each algorithm's hyper-parameters before training.

5.1 CyberBattleSim

The simulation of the network topologies used in this work leverages CyberBattleSim, a tool created by Microsoft, designed to provide an abstraction of communication networks with planted vulnerabilities appropriate for RL training and the MDP framework. The tool supports the node attributes highlighted in Table 2, and the modifications explained in Section 4 have been embedded to support our MDP re-formulation. Three different protocols have been used for the generation of the communication graphs, in particular the Remote Desktop Protocol (RDP), the Server Message Block (SMB), and the Hypertext Transfer Protocol (HTTP). HTTP works as a dummy protocol in this environment, meaning that no credentials or vulnerabilities associated with HTTP will be revealed, making it functionally irrelevant. The inclusion of HTTP is designed to test if the agent can learn to disregard actions associated with redundant outcomes. Regarding the vulnerability assignment, the range of probabilities used for assigning vulnerabilities, alongside the ones for password invalidity, reuse, and firewall configurations, are detailed in Appendix A.1. Based on the sampled probabilities, the generated vulnerabilities will be as follows:

- *ScanWindowsCredentialManagerForRDP*: This local vulnerability allows the agent to search for RDP credentials within the Windows Credential Manager on the node. Exploiting this vulnerability exposes the node IDs and passwords on the RDP port of neighboring RDP nodes, in proportion to *cached rdp password probability*.
- *ScanWindowsCredentialManagerForSMB*: This local vulnerability enables the agent to search for SMB credentials in the Windows Credential Manager of the node. Utilizing this vulnerability reveals the node IDs and passwords on the SMB port of neighboring SMB nodes, according to *cached smb password probability*.
- *ScanWindowsExplorerRecentFiles*: This local vulnerability allows the agent to identify network shares of a node by examining Windows Explorer's recent files. This action leaks only the node IDs of SMB neighbors, with the quantity based on *cached accessed network shares probability*.

- *Traceroute*: This remote vulnerability enables the agent to attempt to discover network nodes using Traceroute. Exploiting this vulnerability exposes the node IDs of RDP and SMB neighbors, proportional to *traceroute discovery probability*.

Some of the planted vulnerabilities may overlap or become irrelevant due to factors like password reuse, invalidity, and the implementation of firewalls. The redesigned action space now includes three local vulnerabilities—*ScanWindowsCredentialManagerForRDP*, *ScanWindowsCredentialManagerForSMB*, and *ScanWindowsExplorerRecentFiles*— the single *Traceroute* remote vulnerability, along with three port connection options—*RDP*, *SMB*, and *HTTP* ports—and the switching actions to move the anchor nodes.

5.2 Selected DRL Algorithms

This study involves some of the main DRL algorithms known in the literature: DQN, PPO, TRPO, and A2C. This choice has been considered to have at least one representative from each class of model-free methods. We also study the behavior of Recurrent PPO (RPPO) [43], an extension of PPO embedding memory capabilities via Long-Short Term Memory (LSTM) layers [22], providing the agent with the capability of remembering the history of sequence steps to determine whether this could improve the agent capabilities. Quantile Regression Deep Q-Network (QRDQN) [11], an extension of the DQN method, has also been considered in the study. This algorithm focuses on predicting a distribution of Q-values for each action versus the approach taken by a traditional DQN that estimates the expected Q-value directly. Researchers have shown that distributional approaches to RL, such as QRDQN, can lead to better convergence properties compared to methods that only estimate expected values [6].

5.3 Neural Network Architecture

All DRL algorithms considered in this study adhere to the pipeline mentioned in Section 4.4 and utilize a consistent NN architecture. This uniformity ensures that differences in performance among the algorithms are not attributable to disparities in the expressiveness of the NN architectures employed. The selected NN architecture contains three layers, arranged in descending order of [256, 128, 64] neurons to decrease the dimensionality of learned features, employing a *LeakyReLU* activation function at each layer for ensuring non-linearity. Training will be conducted using the *Adam* optimizer, utilizing its default hyper-parameters of the SB3 library for optimization. Considering the two extended algorithms, RPPO will incorporate an extra LSTM layer with 32 neurons into the NN architecture, allowing its memory capabilities. On the other hand, QRDQN will model the distribution by employing 200 quantiles, adhering to the default setting in SB3.

5.4 Hyper-parameters Selection

The selection of hyper-parameters for each algorithm was guided by an optimization strategy aimed at identifying the most suited values across the key hyper-parameters and their ranges. To ensure an unbiased approach in the hyper-parameters optimization process towards a specific group of connectivity, these parameters were evaluated across a diverse set of 200 network topologies, with connectivity values uniformly spread within the range of [0.2, 1.0]. The optimization goal was to maximize the Area Under the Curve (AUC) of the average reward during validation, a metric chosen for its ability to simultaneously evaluate the model's generalization to unseen data and its convergence efficiency. More details of the hyper-parameters optimization process are present in Appendix A.3.

6 EXPERIMENTAL RESULTS

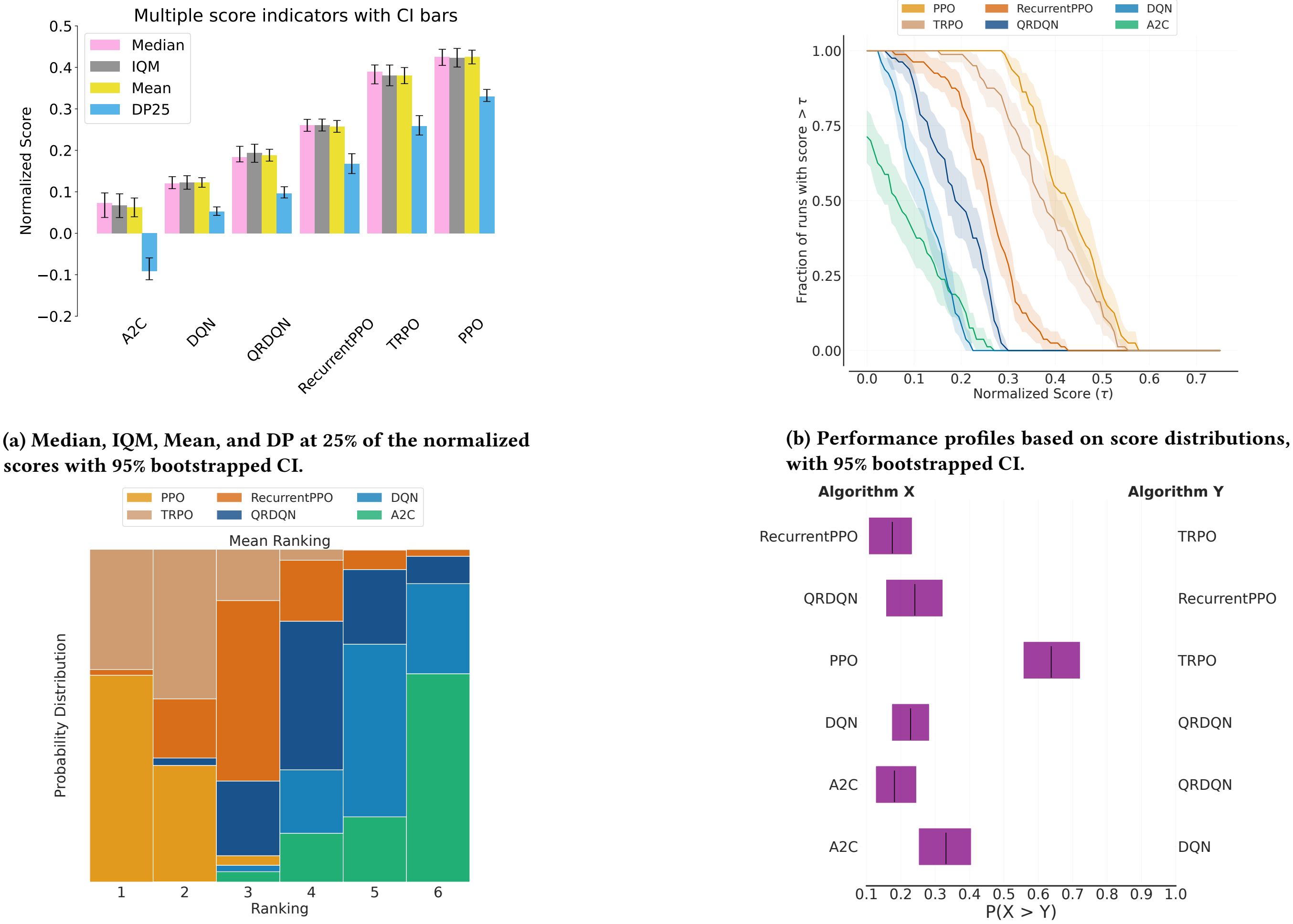
This section presents the experimental results, obtained by aggregating the outcomes across all four groups of topologies, or experiments, to facilitate a comparative analysis of the algorithms regardless of the difficulty level of the topologies. Aggregating performances across all experiments allows us to compare algorithmic performances irrespective of the probabilities of vulnerabilities encountered, thus providing a more generalized view of average performances. Subsequently, the DRL algorithm demonstrating the highest likelihood of superior performance is selected to determine the best training group and to simulate the role of a cyber-attacker.

6.1 Comparative Analysis of DRL Algorithms

This section combines the outcomes from the 20 runs conducted on each of the experiments. In particular, each algorithm underwent training across 20 runs for each experiment, with every run extending over 250,000 iterations. Episodes within these runs had a cut-off at 500 iterations or concluded prematurely if meeting the termination criteria. For each RL algorithm under consideration, we will then rely on 4x20 final summary metrics' curves. We compare results using indicators from the RLiable library, which enhances the reliability of metrics in scenarios with a limited number of runs. Several papers [2] [41] have demonstrated the superiority of aggregating performance metrics using stratified bootstrap CIs, over traditional methods such as standard deviation during RL comparison. By employing bootstrapping, which involves resampling with replacement, we can more accurately estimate the true distribution of performance metrics. For this reason, we utilize 95% stratified bootstrap CIs with 10,000 resamples for all the subsequent indicators.

6.1.1 *Unified evaluation metric.*

Given the variance in scoring scales, like the scale of the average reward, across different groups of topologies—attributable to differences in factors such as the number of discoverable or ownable nodes—we adopt a unified metric for comparing all RL algorithms averaging across all experiments. This metric is the average percentage of nodes owned relative to those reachable from the starting node at each episode, with the AUC of this metric on the validation set used as a key point of comparison. This metric can be obtained from the access graph of each topology and provides a relative performance measure better suited for aggregation, reducing biases associated with using a simple average percentage of owned nodes. Such biases might arise, for example, from the ease with which groups with higher connectivity, can achieve higher scores due to a greater spread of vulnerabilities and more nodes being reachable. Using this metric on the validation set allows us to consider the generalization capabilities of each resulting agent, and the AUC takes into consideration the overall evolution of this metric during training. Furthermore, this score has been normalized using min-max normalization against two



(a) Median, IQM, Mean, and DP at 25% of the normalized scores with 95% bootstrapped CI.

(b) Performance profiles based on score distributions, with 95% bootstrapped CI.

(c) Rank probability distribution for the RL methods compared in the study.

(d) Probability of improvement for each pair of algorithms with overlapping CI of the performance profiles.

Figure 5: RLiable metrics for the comparison of DRL algorithms averaging the results on the four groups of topologies.

benchmarks: the minimum performance of a random agent and the optimal performance value of 100%, which corresponds to the maximum number of nodes ownable from the starter node of the episode.

6.1.2 Comparison results. To enhance the robustness of our conclusions, we report several statistical interval estimates of the score chosen (Figure 5a), including the median, mean, inter-quartile mean (IQM)—which excludes the top and bottom 25% of runs—and an indicator assessing the difficulty progression (DP25), which focuses on the mean scores of the lowest 25% of runs. Employing these diverse indicators mitigates the disadvantages and biases of each of the aggregation measures. Figure 5b shows the performance profiles of the normalized score distributions, showcasing the proportion of runs exceeding specific normalized scores. These profiles offer an unbiased estimator of performance variability and can indicate dominance between methods when one curve consistently lies above another. However, intersecting curves or CIs at various points among RL algorithms suggest that dominance is not absolute and varies across different performance thresholds. To facilitate a deeper understanding of algorithm comparisons, rank plots in Figure 5c visualize the probability for each method to achieve a given rank across all experiments. Using this method shifts the typical approach of binary comparisons by reflecting the probabilistic nature of algorithms' comparison. Additionally, we present results based on the Mann-Whitney U statistic [35] (Figure 5d) that depict the likelihood of one algorithm to overcome another, for each pair of algorithms with overlapping CI of the performance profiles, which may have a non-trivial understanding of their relative difference.

6.1.3 Comparison Highlights. The results demonstrate that policy-based methods generally achieve superior performance on conquering accessible devices across all the indicators, also dominating the probability space for the top three rankings. On the other hand, A2C shows greater variability, particularly between the DP25 and IQM indicators, with the DP25 indicator also showing that more than 25%

of the runs achieve a score below the threshold of a random agent, highlighting the inconsistency in performance across different runs. The superiority of policy-based methods likely comes from the probabilistic nature of the environment, where value-based methods, which tend to converge to deterministic policies, struggle to adapt. Specifically, PPO emerges as the front-runner, with TRPO and RPPO following behind. The inclusion of historical data achieved by RPPO, which uses memory capabilities to remember previous hacking attempts between source-target pairs, seems to deteriorate the performances of the PPO algorithm. Among the value-based approaches, enhancing the DQN agent with the capability to estimate the full distribution of Q values via QRDQN, improves its performance in navigating the uncertain and probabilistic aspects of the environment.

6.2 Exploring the Front-runner Algorithm

The PPO method has been depicted as the front-runner algorithm, dominating the majority of the probability space for the top rank (see Figure 5c). It also demonstrates a superior likelihood of outperforming TRPO, its closest competitor, as evidenced by the higher probability of improvement (depicted in Figure 5d). For this reason, PPO has been used in this section for further analysis.

6.2.1 Evaluating training and testing groups of topologies. Table 3 highlights the performance of the PPO algorithm when trained on a given group of topologies and tested on the test set of another group of topologies, providing an understanding of whether the best strategy should involve training the agent on topologies with more or less planted vulnerabilities or obstacles. For each group, we use the optimal model—determined via the validation set—to compute performance metrics across the different 20 runs. For each run, these metrics were then averaged across 100 episodes on the test set (of the same group or the one of another group, according to the column), employing a *switch interval* of 5 episodes. PPO stands out for its stability and consistent performance across training groups, with a slight improvement observed when trained on connectivity ranges of [0.4, 0.6], suggesting an optimal performance in this intermediate range.

6.2.2 Final evaluation. Considering the best algorithm and best group suited for training in order to guarantee generalization, a series of metrics are presented in Table 4 to conclude the analysis.

Trajectories of the PPO agent on the test set of the best group suited for training (range [0.4, 0.6]) have been generated for 100 episodes and each run, resulting in 2000 different attack paths. All score distributions are present in the case of episodes of length 500, 1000, and 1500 timesteps, showcasing how the results vary while increasing the budget of actions allowed to the agent in an episode. As the agent is allocated more steps, there is a noticeable increase in the discovery of attack paths (defined as the edges of the access graph) and the percentage of nodes owned relative to the maximum possible. However, this expansion comes at the cost of a reduced length optimality factor (minimum actions needed to reach the owned nodes versus the actual actions executed), indicating a less efficient utilization of steps over time. Additionally, the frequency at which the agent revisits source and target nodes increases with a larger step budget, suggesting more repetitive exploration in longer games, especially considering the re-utilization of source nodes.

7 PRACTICAL IMPLICATIONS AND LIMITATIONS

In this section, we outline a practical pipeline designed for network analysts to extract actionable insights from the results produced by the DRL agent execution on a topology. We also address and discuss the current limitations of the pipeline.

7.1 Agent Deployment Pipeline

Following the delineation of the training and evaluation pipelines of the DRL agent (Section 4.4), the experimental section (Section 5 and Section 6) has illustrated a comprehensive evaluation process. This process enables us to identify the optimal training strategy, resulting in a NN that is ready for deployment. The deployment process can be divided into the following steps:

(1) **Vulnerability and topology discovery using scanning tools**: Network analysts employ scanning tools such as Nmap [34] for service scanning, OpenVAS [18] for vulnerability scanning, and a firewall analyzer for firewall scanning. This process results in a graph representation of the topology, where nodes are populated with features extracted from these tools and encoded as detailed in Section 4.2.
(2) **Topology mapping**: Network analysts must map out the network topology, assigning importance values to each node based on how critical it is. The values indicated by the analyst should then be embedded into the node feature vectors of the graph, encoded as detailed in Section 4.2.
(3) **Agent execution**: The agent should be sequentially executed across the entire graph of the topology for multiple episodes. These episodes can start from a fixed starter node or a selected set of starter nodes, such as assets with public-facing applications. This process generates critical trajectories as output since the DRL agent has ideally learned to optimize vulnerability selection within the network. Adjusting the number of episodes allows for balancing statistical significance with execution time efficiency.
(4) **Statistical analysis**: After generating critical trajectories, statistical analysis is applied to unlock insights for the network analyst. While the research community can innovate with customized approaches, we propose several options to gain insights:
    - **Vulnerability-level frequency analysis**: Examining the frequency of vulnerability selections to identify the most critical vulnerabilities. This aids analysts in prioritizing vulnerabilities for patching or reallocating services within the network topology.
    - **Node-level frequency analysis**: Focusing on nodes frequently traversed by attack paths to refine firewall and Intrusion Detection Systems (IDS) [47] rules.
(5) **Optional re-iteration**: The pipeline can be iteratively rerun after applying defense mechanisms that modify the topology. Our tool can then serve as an evaluator of these defenses by measuring how the agent's performance deteriorates after their application.

Table 3: Mean and 95% bootstrapped CI of the average percentage of nodes owned among the reachable ones by the PPO agent in 500 steps. Scores are computed across 20 runs when trained on a group's training set and tested on another group's test set. Bold numbers indicate the top-performing training group for each testing group.

Training/Test groups	0.2-0.4	0.4-0.6	0.6-0.8	0.8-1.0	Average
0.2-0.4	**0.566 ± 0.016**	0.556 ± 0.015	0.607 ± 0.013	0.599 ± 0.012	*0.582 ± 0.014*
0.4-0.6	0.552 ± 0.017	**0.578 ± 0.015**	**0.614 ± 0.014**	0.645 ± 0.011	**0.597 ± 0.013**
0.6-0.8	0.536 ± 0.016	0.513 ± 0.015	0.608 ± 0.013	0.641 ± 0.012	*0.575 ± 0.014*
0.8-1.0	0.524 ± 0.016	0.537 ± 0.014	0.603 ± 0.013	**0.665 ± 0.012**	*0.582 ± 0.014*

Table 4: Mean and 95% CI of PPO distribution metrics on the test set with connectivity in the range [0.4, 0.6].

Metric	500 steps	1000 steps	1500 steps
Discovered attack paths	0.613 ± 0.011	0.740 ± 0.009	0.799 ± 0.008
Owned nodes among reachable	0.588 ± 0.015	0.645 ± 0.016	0.687 ± 0.016
Length optimality factor	0.284 ± 0.005	0.214 ± 0.004	0.181 ± 0.004
Source node revisiting ratio	0.385 ± 0.011	0.504 ± 0.013	0.549 ± 0.014
Target node revisiting ratio	0.380 ± 0.010	0.396 ± 0.010	0.403 ± 0.011

7.2 Limitations

The previously proposed pipeline can be implemented using our DRL agent. However, our methodology still faces certain constraints, primarily related to scalability, which may also impact the deployment of the agent:

- **Dependency on the vulnerability set**: The trained DRL agent optimizes the vulnerability selection limited to the vulnerabilities within its action space. As new vulnerabilities arise in the network topology — such as through the introduction of new services into nodes that may bring new vulnerabilities — the agent must be re-trained by integrating these vulnerabilities into its action set.
- **Scalability concerning vulnerabilities**: An increase in the size of the vulnerabilities set can pose scalability challenges. A larger set requires more exploration by the agent to find optimal action sequences and as a consequence, the agent may require more training time and longer episode iterations. To manage this challenge, we propose reducing the re-training time by fine-tuning only the last NN layer.
- **Scalability concerning the local view**: In scenarios involving local source-target node pairs, additional switching actions are required to cover all possible target nodes. Although our model's topology independence enhances versatility in evolving environments, it may result in longer episode iterations on specific larger topologies compared to approaches using a global view tuned to a single topology.

8 CONCLUSIONS

This paper delved into the application of DRL for the training of a cyber-attacker model. The learned policy can be subsequently used for generating attack paths on input topologies, offering valuable insights to network analysts into potential defense strategies.

8.1 Research Highlights

By reformulating the MDP, this work introduces a topology independent NN framework capable of being used on diverse topologies. This is followed by a comprehensive training and evaluation strategy that involves a variety of topologies, each with a different size or planted set of vulnerabilities, to ensure the trained agent's ability to generalize across different scenarios. We provided, in this work, the definition of additional metrics and graph abstractions that assess the relative complexity of topology from the perspective of an RL agent. A comparative analysis of various representative RL algorithms, each fine-tuned with a set of optimized hyper-parameters and a benchmark NN, has been conducted. Results from the utilization of the RLiable library highlight the higher potential of policy-based methods in navigating the stochastic nature of the probabilistic network environments used in this study, compared to other classes of RL methods. A further analysis of PPO reveals its superior capability for generalizing across topologies. The empirical data demonstrates the agent's proficiency in achieving control over a significant portion of the nodes (69% ± 2%) and discovering a large set of possible attack paths (80% ± 1%) on a reserved set of test topologies, when the agent is provided with a limited span of 1500 actions. Training such models can facilitate the development of more robust defense mechanisms by analyzing the strategies taken by a learned attacker policy.

8.2 Future Work

Future directions will involve the extension of the training and evaluation scenarios to encompass a broader, more diverse set of vulnerabilities, and enhancing the simulation by incorporating realistic data. Moreover, we aim to determine more realistic reward functions, which might also be more expressive than a goal-agnostic weighted linear combination of action outcomes.

ACKNOWLEDGMENTS

This work has been partially supported by the French National Research Agency under the France 2030 label (Superviz ANR-22-PECY-0008). The views reflected herein do not necessarily reflect the opinion of the French government.

REFERENCES

[1] Mohamed Abdlhamed, Kashif Kifayat, Qi Shi, and William Hurst. 2017. *Intrusion prediction systems*. Springer, 155–174. https://doi.org/10.1007/978-3-319-44257-0_7

[2] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. 2022. Deep Reinforcement Learning at the Edge of the Statistical Precipice. arXiv:2108.13264 [cs.LG]

[3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. arXiv:1907.10902 [cs.LG]

[4] Andy Applebaum, Camron Dennler, Patrick Dwyer, Marina Moskowitz, Harold Nguyen, Nicole Nichols, Nicole Park, Paul Rachwalski, Frank Rau, Adrian Webster, and Melody Wolk. 2022. Bridging Automated to Autonomous Cyber Defense: Foundational Analysis of Tabular Q-Learning. In *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security* (Los Angeles, CA, USA) *(AISec'22)*. Association for Computing Machinery, New York, NY, USA, 149–159. https://doi.org/10.1145/3560830.3563732

[5] Callum Baillie, Maxwell Standen, Jonathon Schwartz, Michael Docking, David Bowman, and Junae Kim. 2020. CybORG: An Autonomous Cyber Operations Research Gym. arXiv:2002.10667 [cs.CR]

[6] Marc G. Bellemare, Will Dabney, and Rémi Munos. 2017. A Distributional Perspective on Reinforcement Learning. arXiv:1707.06887 [cs.LG]

[7] Aimad Berady, Mathieu Jaume, Valérie Viet Triem Tong, and Gilles Guette. 2022. PWNJUTSU: A Dataset and a Semantics-Driven Approach to Retrace Attack Campaigns. *IEEE Transactions on Network and Service Management* 19, 4 (2022), 5252–5264. https://doi.org/10.1109/TNSM.2022.3183476

[8] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, Vol. 24.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:1606.01540 [cs.LG]

[10] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. 2018. How Many Random Seeds? Statistical Power Analysis in Deep Reinforcement Learning Experiments. arXiv:1806.08295 [cs.LG]

[11] Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. 2017. Distributional Reinforcement Learning with Quantile Regression. arXiv:1710.10044 [cs.AI]

[12] Bradley Efron. 1979. Bootstrap methods: Another look at the jackknife. *Annals of Statistics* 7, 1 (1979), 1–26.

[13] H. Farhadi, M. AmirHaeri, and M. Khansari. 2011. Alert correlation and prediction using data mining and HMM. *The ISC International Journal of Information Security* 3, 2 (2011), 77–101. https://doi.org/10.22042/isecure.2015.3.2.3

[14] Ouissem Ben Fredj, Alaeddine Mihoub, Moez Krichen, Omar Cheikhrouhou, and Abdelouahid Derhab. 2020. CyberSecurity Attack Prediction: A Deep Learning Approach. In *13th International Conference on Security of Information and Networks (SIN 2020)*. ACM, Merkez, Turkey, 6. https://doi.org/10.1145/3433174.3433614

[15] Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3, 4 (1999), 128–135.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[18] Greenbone Networks GmbH. Year. OpenVAS - Open Vulnerability Assessment System. https://www.openvas.org/

[19] Xiaotong Guo, Jing Ren, Jiangong Zheng, Jianxin Liao, Chao Sun, Hongxi Zhu, Tongyu Song, Sheng Wang, and Wei Wang. 2023. Automated Penetration Testing with Fine-Grained Control through Deep Reinforcement Learning. *Journal of Communications and Information Networks* 8, 3 (2023), 212–220. https://doi.org/10.23919/JCIN.2023.10272349

[20] Simon Haykin. 1998. *Neural Networks: A Comprehensive Foundation* (2nd ed.). Prentice Hall PTR, USA.

[21] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2019. Deep Reinforcement Learning that Matters. arXiv:1709.06560 [cs.LG]

[22] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[23] Zhenguo Hu, Razvan Beuran, and Yasuo Tan. 2020. Automated Penetration Testing Using Deep Reinforcement Learning. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 2–10. https://doi.org/10.1109/EuroSPW51379.2020.00010

[24] Zhenguo Hu, Razvan Beuran, and Yasuo Tan. 2020. Automated Penetration Testing Using Deep Reinforcement Learning. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 2–10. https://doi.org/10.1109/EuroSPW51379.2020.00010

[25] Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. 2020. Normalization Techniques in Training DNNs: Methodology, Analysis and Application. arXiv:2009.12836 [cs.LG]

[26] Todd Hughes and Oleg Sheyner. 2003. Attack scenario graphs for computer network threat analysis and prediction. *Complex.* 9, 2 (nov 2003), 15–18. https://doi.org/10.1002/cplx.20001

[27] Martin Husák, Jana Komárková, Elias Bou-Harb, and Pavel Čeleda. 2019. Survey of Attack Projection, Prediction, and Forecasting in Cyber Security. *IEEE Communications Surveys & Tutorials* 21, 1 (2019), 640–660. https://doi.org/10.1109/COMST.2018.2871866

[28] Norman L. Johnson, Samuel Kotz, and N. Balakrishnan. 1995. *Continuous Univariate Distributions, Volume 2.* Wiley.

[29] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101, 1 (1998), 99–134. https://doi.org/10.1016/S0004-3702(98)00023-X

[30] I. Kalderemidis, A. Farao, P. Bountakas, S. Panda, and C. Xenakis. 2022. GTM: Game Theoretic Methodology for optimal cybersecurity defending strategies and investments. In *Proceedings of the 17th International Conference on Availability, Reliability and Security.* ACM, Vienna, Austria, 1–9. https://doi.org/10.1145/3538969.3544431

[31] Li Li, Raed Fayad, and Adrian Taylor. 2021. CyGIL: A Cyber Gym for Training Autonomous Agents over Emulated Network Systems. arXiv:2109.03331 [cs.CR]

[32] Qian Li, Minghao Hu, Hongxia Hao, and et al. 2023. INNES: An intelligent network penetration testing model based on deep reinforcement learning. *Applied Intelligence* 53 (2023), 27110–27127. https://doi.org/10.1007/s10489-023-04946-1

[33] Y. Li, Y. Wang, X. Xiong, J. Zhang, and Q. Yao. 2022. An Intelligent Penetration Test Simulation Environment Construction Method Incorporating Social Engineering Factors. *Applied Sciences* 12, 12 (2022), 6186. https://doi.org/10.3390/app12126186

[34] Gordon Lyon. 2008. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning.* Insecure.Com LLC.

[35] Henry B. Mann and Donald R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *Annals of Mathematical Statistics* 18, 1 (1947), 50–60.

[36] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. arXiv:1602.01783 [cs.LG]

[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[38] Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. 2022. Model-based Reinforcement Learning: A Survey. arXiv:2006.16712 [cs.LG]

[39] James R Norris. 1998. *Markov Chains.* Cambridge University Press.

[40] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. 2005. MulVAL: A Logic-based Network Security Analyzer. In *14th USENIX Security Symposium (USENIX Security 05).* USENIX Association, Baltimore, MD.

[41] Andrew Patterson, Samuel Neumann, Martha White, and Adam White. 2023. Empirical Design in Reinforcement Learning. arXiv:2304.01315 [cs.LG]

[42] Aritran Piplai, Mike Anoruo, Kayode Fasaye, Anupam Joshi, Tim Finin, and Ahmad Ridley. 2022. Knowledge Guided Two-player Reinforcement Learning for Cyber Attacks and Defenses. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA).* 1342–1349. https://doi.org/10.1109/ICMLA55696.2022.00213

[43] Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss. 2022. Generalization, Mayhems and Limits in Recurrent Proximal Policy Optimization. arXiv:2205.11104 [cs.LG]

[44] Martin L. Puterman. 1994. Markov Decision Processes: Discrete Stochastic Dynamic Programming.

[45] X. Qin and W. Lee. 2004. Attack plan recognition and prediction using causal networks. In *20th Annual Computer Security Applications Conference.* 370–379. https://doi.org/10.1109/CSAC.2004.7

[46] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* 22, 268 (2021), 1–8. http://jmlr.org/papers/v22/20-1364.html

[47] Martin Roesch. 1999. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the USENIX LISA '99 Conference.* https://www.usenix.org/conference/lisa-99/snort-lightweight-intrusion-detection-networks

[48] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2017. Trust Region Policy Optimization. arXiv:1502.05477 [cs.LG]

[49] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG]

[50] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press.

[51] Microsoft Defender Research Team. 2021. CyberBattleSim. https://github.com/microsoft/cyberbattlesim. Created by Christian Seifert, Michael Betser, William Blum, James Bono, Kate Farris, Emily Goren, Justin Grana, Kristian Holsheimer, Brandon Marken, Joshua Neil, Nicole Nichols, Jugal Parikh, Haoran Wei..

[52] Franco Terranova, Abdelkader Lahmadi, and Isabelle Chrisment. 2024. Code for the paper "Leveraging Deep Reinforcement Learning for Cyber-Attack Paths

Prediction: Formulation, Generalization, and Evaluation". https://doi.org/10.5281/zenodo.12783985. https://doi.org/10.5281/zenodo.12783985

[53] Franco Terranova, Abdelkader Lahmadi, and Isabelle Chrisment. 2024. Topologies, Checkpoints, and Configurations for the paper "Leveraging Deep Reinforcement Learning for Cyber-Attack Paths Prediction: Formulation, Generalization, and Evaluation". https://doi.org/10.5281/zenodo.12783622. https://doi.org/10.5281/zenodo.12783622

[54] Junkai Yi and Xiaoyan Liu. 2023. Deep Reinforcement Learning for Intelligent Penetration Testing Path Design. *Applied Sciences* 13, 16 (2023). https://doi.org/10.3390/app13169467

[55] Yue Zhang, Jingju Liu, Shicheng Zhou, Dongdong Hou, Xiaofeng Zhong, and Canju Lu. 2022. Improved Deep Recurrent Q-Network of POMDPs for Automated Penetration Testing. *Applied Sciences* 12, 20 (2022). https://doi.org/10.3390/app122010339

A APPENDIX

A.1 Topology Generation

Environment generation using the CyberBattleSim tool was carried out with the creation of several topologies with hyper-parameters varying in proper ranges. The vulnerabilities integrated within each simulated network environment are determined according to the probabilities outlined in Table 6. These probabilities cover various aspects, including specific network protocols, firewall configurations, credential reuse across different nodes, and the potential for previously valid passwords to become invalid. For each topology, the described probabilities along with other relevant hyper-parameters were systematically sampled using a random sampling method. The sampling ranges were defined as detailed in Table 5, ensuring a diverse set of configurations to robustly test the network simulations under varied conditions.

Table 5: Topology parameters varied in order to generate topologies and the corresponding range used.

Parameter	Range
Number of Clients	[40, 70]
Number of Servers	[10, 30]
α of the inter-beta distribution	[1, 200]
β of the inter-beta distribution	[1000, 5000]
α of the intra-beta distribution	[1, 200]
β of the intra-beta distribution	[1000, 5000]
cached_rdp_password_probability	[0.4, 0.8]
cached_smb_password_probability	[0.4, 0.8]
cached_accessed_network_shares _probability	[0.2, 0.5]
traceroute_discovery_probability	[0.3, 0.7]
cached_password_has_changed _probability	[0.01, 0.1]
probability_two_nodes_use _same_password_to_access _given_resource	[0.1, 0.33]
firewall_rule_incoming_probability	[0.0, 0.2]
firewall_rule_outgoing_probability	[0.0, 0.2]

A.2 Prioritization Coefficients

Considering the reward function outlined in Section 4.1.3, the coefficients for the linear combination were adjusted to prioritize node control over information gathering. To achieve this balance, taking into account the scale of values influenced, the coefficients within the linear combination of the reward function were carefully chosen as outlined below:

Table 6: Descriptions of the probabilities of vulnerability assignment used in CyberBattleSim.

Name	Description	Outcome
cached_rdp_ password_ probability	Probability of RDP credential leakage.	Node ID and password leaked to RDP neighbor.
cached_smb_ password_ probability	Probability of SMB credential leakage.	Node ID and password leaked to SMB neighbor.
cached_accessed_ network_ shares_probability	Probability of node ID leakage via SMB.	Node ID leaked to SMB neighbor.
traceroute_ discovery_ probability	Probability of node ID leakage via SMB or RDP.	Node ID leaked by SMB or RDP neighbor.
cached_password_ has_ changed_probability	Probability that a previously leaked password has changed.	Leaked password becomes invalid for port connection.
probability_two_ nodes_use_ same_password_ to_access_ given_resource	Probability of credential re-use among nodes' ports.	Credential valid for multiple nodes.
firewall_rule_ incoming_ probability	Probability of a BLOCK rule on incoming ports.	BLOCK rule applied to each incoming listening port.
firewall_rule_ outgoing_ probability	Probability of a BLOCK rule on outgoing ports.	BLOCK rule applied to each outgoing port.

$$
\begin{aligned}
K_{\text{value}} &= 5 \\
K_{\text{node discover}} &= 5 \\
K_{\text{credential discover}} &= 3 \\
K_{\text{cost}} &= 1 \\
K_{\text{first success}} &= 7
\end{aligned}
$$

These coefficients determine a reward function scale that redirects the behavior of the RL agents, encouraging them to adopt strategies that improve information gathering, but with the primary goal of enhancing overall control over network nodes based on a general strategy rather than targeting specific nodes. Additionally, operational costs are associated with the actions, which are set to the default values provided by CyberBattleSim. Additionally, in the computation of the connectivity metric used to assess the complexity of a topology (Section 4.3.3), the prioritization assigned as well greater importance to ease of access over ease of discovery within the network. The assigned coefficients have been $\alpha = 0.75$ for the accessibility term and $\beta = 0.25$ for the discovery term in the metric. This assignment results in higher connectivity values for topologies that are more accessible than discoverable, emphasizing as well ease of access as a priority over the ease of discovery within the network.

A.3 Hyper-parameters Optimization

The optimization of RL algorithms' hyper-parameters aimed at maximizing the AUC of the average reward on the validation set. We executed 50 trials per algorithm, with each trial consisting of three runs under consistent seed settings. These trials utilized the Tree-structured Parzen Estimator [8] algorithm implemented via the Optuna framework [3]. The hyper-parameters search spaces are delineated in Table 7. To simplify the process, hyper-parameters optimization was conducted on the fundamental versions of the algorithms, and the derived hyper-parameters were subsequently applied as well to their more sophisticated counterparts, RPPO and QRDQN, ensuring also fairness when comparing each fundamental and sophisticated version. The optimal hyper-parameter configurations for each algorithm will be documented in a *yaml* file situated in the logs folder (attached to the paper) corresponding to each algorithm.

Table 7: Hyper-parameters' search space across DRL algorithms including general and model-specific parameters.

Hyper-parameter	**Values**
Learning Rate	[0.0001, 0.001, 0.01]
Batch Size	[64, 128, 256]
Discount Factor (γ)	[0.9, 0.95, 0.99]
Experience Replay Buffer Size (DQN)	[5000, 10000, 20000]
Tau τ (DQN)	[0.1, 0.5, 1.0]
Policy Clip Range (PPO)	[0.1, 0.2, 0.3]
Number of Steps per Update (PPO, TRPO)	[1024, 2048, 4096]
Number of Steps per Update (A2C)	[5, 10, 20]
Maximum Gradient Norm (PPO, A2C)	[0.3, 0.5, 0.7]
Entropy Coefficient (PPO, A2C)	[0.01, 0.1, 0.2]
Value Function Coefficient (PPO, A2C)	[0.5, 1.0, 1.5]
Target KL Divergence (TRPO)	[0.01, 0.05, 0.1]
GAE Lambda (TRPO, A2C)	[0.9, 0.95, 0.99]