Accelerating Autonomous Cyber Operations: A Symbolic Logic Planner Guided Reinforcement Learning Approach

Ryan Kerr Steven Ding
Queen's School of Computing
Kingston, Ontario, Canada
{ryan.kerr, steven.ding}@queensu.ca

Li Li Adrian Taylor
Defence Research and Development Canada
Ottawa, Ontario, Canada
li.li, adrian.taylor@drdc-rddc.gc.ca

***Abstract*—Training a reinforcement learning agent to learn network penetration testing is challenging due to the partially-observable, non-deterministic environment. The large action space leads to extended training time, an issue of particular concern in mission-oriented network deployment that requires timely hardening tests. Current solutions for automating penetration testing are divided between reinforcement learning (RL) and AI planning. This work integrates the two paradigms and establishes a neuro-symbolic agent training system through an interactive symbolic logic engine. Two methods are examined for accelerating the pentest agent training in this system, namely: invalid action masking for Deep Q-Networks and using a symbolic logic engine as an environment driver. The results show that invalid action masking is highly effective at reducing the number of steps to convergence, while the logic-based simulator provides a significant per-step performance improvement to speed up training. These results highlight that a hybrid neuro-symbolic approach is a viable, and perhaps even necessary, method for developing and improving cyber RL agents.**

I. Introduction

As the world becomes increasingly interconnected, cybersecurity is critical for protecting our modern way of life. However, cybersecurity is unfortunately asymmetrical by its very nature: defenders must maintain full coverage over their networks while adversaries need only a handful of flaws to engender considerable harm and devastating consequences. This divide is further exacerbated by the global shortage of cyber security talent, motivating autonomous cyber defense capabilities to fill in the gap. The development of a fully autonomous defensive (blue) agent is critical for levelling the playing field. Many tools for automating or simplifying defense tasks are available today.

Thus, much of the current research is directed toward developing adversarial (red) agents that are capable of carrying out simulated attacks against one's own network. This practice, known as penetration testing, is used by human experts to identify vulnerabilities in a network's defences, provide actionable steps towards hardening the network against a real attack. Penetration testing is also commonly used as a method for training cybersecurity experts. However, red team exercises, such as penetration testing, are costly in terms of both the expertise and the time required to execute. An autonomous red agent allows defenders to harden their networks and get ahead of emergent threats at a considerably reduced cost.

Prior work in simulated penetration testing has been largely divided into two categories: domain knowledge-driven approaches and *Deep Reinforcement Learning* (DRL) approaches. Knowledge-driven approaches rely on the ability to formally define the dynamics of an environment – often in the form of symbolic logic [1, 2, 3] – to deterministically derive how an agent should act. Recent work by Miller et al. [4] applied a variant of online re-planning whereby an agent constructs a partial plan based on its current knowledge, executes the plan in the environment, and then repeats the process, incorporating any new observations, until the goal is reached. Since the agent may not be able to sufficiently plan ahead with limited visibility, Miller et al. [4] implemented rules for extrapolating from the current state and only executed the first action of each plan before repeating the full process.

The opposite branch, DRL, does not require complete knowledge of the environment and instead relies on experience to determine how an agent will act. Agents try to learn the best course of action purely through repeated interaction with the environment. Since a formal model is not required for the agent, researchers have developed and trained agents on everything from abstract simulations of networks [5, 6, 7] to real networks [8, 9, 10] with the hopes discovering novel tactics. However, a key drawback of RL is that the agent must learn the environment dynamics from scratch, and as a consequence, take upwards of millions of interactions to show any meaningful improvement – especially when faced with large action spaces [7].

The purpose of the research presented in this paper is to determine the viability of a middle ground where an agent can exploit domain knowledge to accelerate learning while retaining the capacity to explore and demonstrate emergent behaviour. To this end, this paper presents the following contributions:

- AR1ST0TLE, an interactive PDDL symbolic logic engine.
- A novel use of PDDL for invalid action masking.
- The use of a symbolic logic engine as an RL training environment.

979-8-3503-7099-7/24/$31.00 ©2024 Crown

II. Problem Description

A. Partially Observable Markov Decision Process (POMDP)

Markov decision Processes (MDPs) are a mathematical framework for describing decision problems in domains with uncertainty. An MDP is formally defined by the 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}$ is a transition function giving the probability of reaching a successor state $s' \in \mathcal{S}$ from a given state-action pair $\mathcal{T}(s, a) = Pr(s'|s, a)$, $\mathcal{R}$ is a reward function $\mathcal{R}(s, a, s')$.

The agent's overall objective is to learn a policy π for selecting the best action a for a given state s, such that the agent maximizes its accumulated reward.

Partially-observable MDPs (POMDPs) are a generalization over MDPs where an agent only has partial visibility of the environment. Given an agent only has partial visibility of the environment, the MDP is extended to a POMDP by including an observation space Ω and an observation function $\mathcal{O}$ that gives the probability of getting an observation o when moving to a new state $\mathcal{O}(s', a, o) = Pr(o|s', a)$.

In a POMDP, the agent operates on a probability distribution over $\mathcal{S}$ rather than on ground states $s \in \mathcal{S}$ directly. These distributions, referred to as belief states b, reflect the agent's certainty of being in, or not in, any of the underlying states s. When taking actions, the agent updates its belief states based on the observations it receives. By operating at this higher level, it is possible to solve a POMDP by treating it as an MDP over belief states rather than ground states.

B. Classical Planning

Planning problems are defined by $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ where $\mathcal{F}$ is the set of atoms, $\mathcal{A}$ is the action space, $\mathcal{I}$ is the initial state where $\mathcal{I} \subset \mathcal{F}$, and $\mathcal{G}$ is the goal condition. A state can be described as the subset of atoms in $\mathcal{F}$ that are $true$, and by the closed-world assumption, atoms not found in s are implied $false$. Every action $a \in \mathcal{A}$ can be further divided into the tuple $\langle \mathrm{pre}_a, \mathrm{eff}_a \rangle$ where pre_a describes the action's precondition and eff_a is the action's effect. An action a is said to be applicable[1] for a given state s iff pre_a holds in s. This can be thought of as an MDP where $\mathcal{T}(s, a, s') = 1$ if an action is valid and 0 otherwise.

The objective of a planner is to find a sequence of actions $\Pi = \langle a_0, a_1, \ldots, a_n \rangle$ that takes the agent from the initial state $\mathcal{I}$ to some goal state $\mathcal{S}_\mathcal{G}$ while optionally maximizing some utility function, such as an expected reward, or minimizing a cost, such as the plan length.

In practice, planners use the *Planning Domain Definition Language* (PDDL) to describe problem domains and divide the problem into two parts: the domain, describing the dynamics of the environment, and a problem, describing a specific instance. Domain definitions specify predicates, which are possible properties of objects that can be true or false, and operators, which serve as templates for actions by defining parameterized versions of pre_a and eff_a. The problem definition declares the specific objects within a domain instance, such as the hosts on a network and their software, in addition to the initial state $\mathcal{I}$ and the goal $\mathcal{G}$. The combination of the predicates and objects form $\mathcal{F}$. For instance, the predicate `has_software` can be combined with the host object `WebServer` and a software object `NGINX` to a possible fact (has_software WebServer NGINX) $\in \mathcal{F}$. The same reasoning applies to operators and $\mathcal{A}$: when an operator, such as `Escalate`, is combined with the an appropriate object, such as the earlier server, it forms the action (Escalate WebServer) $\in \mathcal{A}$.

[1]The terms (in)applicable and (in)valid are used interchangeably throughout this paper.

C. Online Planning Problems

As the agent has partial observability and no prior knowledge of the environment, it requires an analogue of POMDPs for planning problems. As such, the agent's problem $\mathcal{P}_k$ is described as a dynamic subset of the full problem $\mathcal{P}$ (i.e. $\mathcal{P}_k \subseteq \mathcal{P}$) whereby $\mathcal{P}_k = \langle \mathcal{F}_k, \mathcal{A}_k, \mathcal{I}_k, \mathcal{G}_k \rangle$ and k denotes the subset of known elements in the corresponding set (i.e. $\mathcal{F}_k \subseteq \mathcal{F}$ and so on). The known action space $\mathcal{A}_k$ is further defined as $\{a \in \mathcal{A} : \mathrm{pre}_a \setminus \mathcal{F}_k = \emptyset\}$. Since the agent does not know the values (let alone the existence) of $f \notin \mathcal{F}_k$, the agent's ability to reason about action effects is bounded by $\mathcal{F}_k$ (i.e. $\mathrm{eff}_{ak} = \mathrm{eff}_a \cap \mathcal{F}_k$). The true effect eff_a of an action a is only revealed after running the action, at which point the agent can update its knowledge to include previously unknown facts $\mathcal{F}_k \leftarrow \mathcal{F}_k \cup \mathrm{eff}_a$. This method is described as "online" as the full plan cannot be constructed prior to executing actions due to the necessary interaction with the environment and considers the possibility that external factors may invalidate the execution of a plan.

D. Deep Reinforcement Learning (DRL)

Reinforcement Learning (RL) is a method whereby an agent learns to maximize a reward signal through repeated interaction with an environment. Critically, RL agents are capable of learning without having any knowledge of the environment dynamics, including the transition probabilities $\mathcal{T}$, action preconditions pre_a, and action effects eff_a. This capability enables agents to learn an optimal policy in complex environments, such as a live network.

A red agent typically begins with initial access on a host and must navigate through the network to achieve its goal of impacting a target host. The red agent learns an optimized attack course of action without any domain-specific knowledge through repeated trial-and-error alone. For this paper, this is achieved using the current state-of-the-art algorithm, *Proximal Policy Optimization* (PPO), which learns a policy function $\pi : \mathcal{S} \rightarrow \mathcal{A}$ to select the best action $a \in \mathcal{A}$ for a given state $s \in \mathcal{S}$ based on experience.

III. The Symbolic Logic Engine

To improve the performance of an autonomous red agent, the proposed solution integrates an online planning logic engine with DRL. AR1ST0TLE is the devised PDDL1.2-compatible Haskell logic engine designed to function in environments

with minimal prior knowledge and unbounded uncertainty. AR1ST0TLE supports a number of PDDL extensions, including the ADL extension set, allowing for increased expressiveness while still maintaining compatibility with existing off-the-shell planners, such as *Fast-Forward*[2] (FF) [11].

The primary use-case of AR1ST0TLE is to inform a decision-making agent of the actions that are applicable, or valid, for a given state ($\mathcal{A}_V = \{a \in \mathcal{A}_k : pre_a(s)\}$). Similar to online re-planning frameworks, AR1ST0TLE is designed to be interleaved with every step of execution in order to monitor changes to the environment's state. At each step, the agent selects an action from the list of applicable actions, executes the action in the environment, and then reports the resulting observations to AR1ST0TLE as a conjunction of PDDL-formatted atoms. Considering the agent is operating over the partially-observable problem, $\mathcal{P}_k$, rather than the fully observable problem, $\mathcal{P}$, the agent will inevitably discover previously unknown objects ($f \notin \mathcal{F}_k$) as it navigates through the environment. When an unknown object is identified in the observation, AR1ST0TLE gleans the type information from the corresponding predicate definition and registers the new object with the appropriate type, expanding $\mathcal{F}_k$. Once all objects and their types have been resolved, AR1ST0TLE can optionally combine the expected action effect with the observation, then evaluate each operator to generate the list of applicable actions to return to the agent.

This use-case is in contrast to existing uncertainty-aware symbolic logic systems, such as Bonet and Geffner [12]'s K-replanner and Miller et al. [4]'s planning system, where the system serves as the decision making controller rather than as an auxiliary system. Unlike the implementation by Miller et al. [4], AR1ST0TLE is entirely PDDL-based similar to the K-replanner. However, unlike the K-replanner, AR1ST0TLE does not share the requirement to explicitly declare facts as "hidden" a priori and does not place restrictions on what can appear in the effect of an action (with the exception of contradictions), allowing for a greater degree of flexibility.

IV. Experiment Environment

The *Cyber Operations Research Gym* (CybORG)[3] is an open-source, OpenAI gym-compatible penetration testing simulator and emulator for training RL agents to attack or defend a network [13]. CybORG includes a number of training scenarios of which scenario 1b is selected as an experimental example in this paper. Scenario 1b features a small enterprise network composed of three subnets: a user subnet containing five hosts, an enterprise subnet with four hosts, and an operational subnet with an additional four hosts. All hosts within a subnet are capable of communicating directly with each other, but only certain hosts can communicate with specific hosts on a remote subnet.

The red agent begins on a specific host in the user subnet. The goal of the red agent is to impact a service on a specific server located in the operational subnet. To achieve this, the agent has the following five operations at its disposal:

- **NetScan (Subnet)**: Reveals all hosts on the specified subnet. Requires the agent to have superuser access to at least one of the hosts on the specified subnets.
- **PortScan (Host)**: Reveals all listening ports open on a specified host. Requires the agent to have a foothold on the host's subnet and have visibility of the host.
- **Exploit (Host)**: Attempts to exploit a remote service on a host, giving the agent basic (user) access of the host.
- **Escalate (Host)**: Attempts a privilege escalation to upgrade the agent's access level on the specified host from basic access to superuser access. This action will additionally reveal any subnet-external hosts that are reachable from the target host.
- **Impact (Host)**: Impacts the function/services on the specified host. Requires superuser access.

When enumerated with all the possible parameters, such as the specific host or subnet to target, the agent can take a total of 55 different actions in the scenario environment. The original CybORG scenaio features a defender (blue) agent that attempts to mitigate the red agent's advances and remediate any compromised hosts, however, the defender is disabled for the purpose of this research. In this work, red actions are monotonic, meaning past observations and progress cannot be erased or invalidated, remaining consistent with prior literature [14, 4].

1) NetScan UserNet	9) PortScan Enterprise3
2) PortScan User$\{1^a,2^b,3^c,4^d\}$	10) Exploit Enterprise3
3) Exploit User$\{1^a,2^b,3^c,4^d\}$	11) Escalate Enterprise3
4) Escalate User$\{1^a,2^b,3^c,4^d\}$	12) PortScan OpServer
5) PortScan Enterprise$\{0^{a,b},1^{c,d}\}$	13) Exploit OpServer
6) Exploit Enterprise$\{0^{a,b},1^{c,d}\}$	14) Escalate OpServer
7) Escalate Enterprise$\{0^{a,b},1^{c,d}\}$	15) Impact OpServer
8) NetScan EnterpriseSubnet	

TABLE I: The optimal course of action for the red agent. There are 4 redundant paths, marked by the superscripts a-d.

The shortest – and optimal – attack path for the agent is as follows: the agent performs "NetScan" on the initial subnet, revealing the four neighbouring hosts. The agent then performs the sequence "PortScan" followed by "Exploit" then "Escalate" against any one of the newly discovered hosts to reveal a new host located in the intermediate, enterprise subnet. The agent then repeats the same sequence against the enterprise host to gain a foothold on the enterprise subnet, where the agent must repeat the process against an enterprise host. Unlike the user subnet, there is only a single host that leads to the operational subnet from the enterprise subnet. Assuming the agent successfully selects the correct host, the agent will gain superuser access to the target server after carrying out the above steps. At this point, the agent can use the "Impact" action against the operation server, achieving its goal, and ending the scenario. In total, this path consists of 15 steps.

Two reward functions are applied in the experiments:

[2]Available from https://fai.cs.uni-saarland.de/hoffmann/ff.html

[3]Available from https://github.com/cage-challenge/CybORG

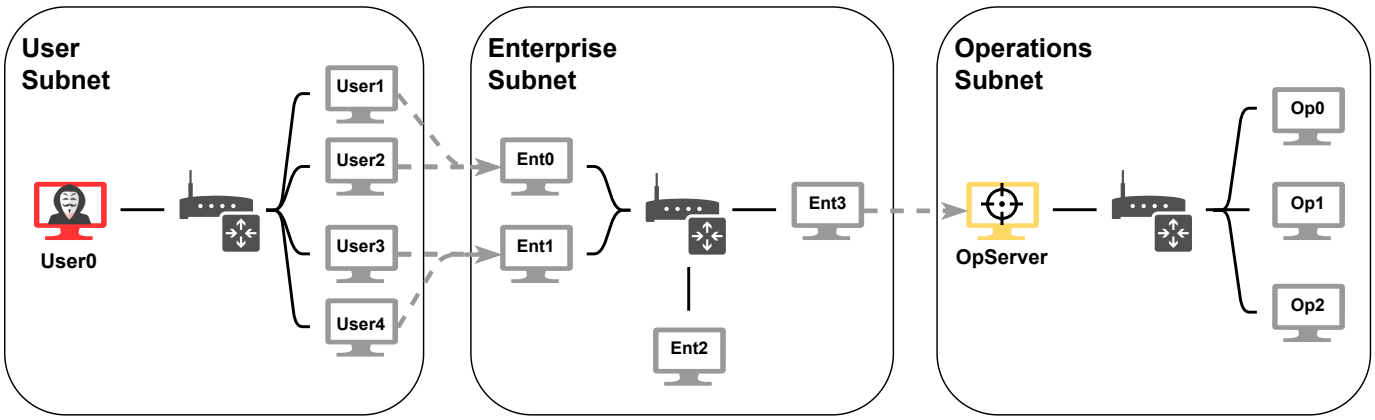

Fig. 1: Scenario diagram. Each box demarks a subnet where all hosts within the subnet can communicate with each other. The dashed arrows indicate inter-subnet reachability. The adversary (red) begins in the user subnet and must progressively scan for and exploit hosts to find its way to impact the goal host (yellow).

1) The agent is only rewarded upon reaching its goal.
2) The agent is given a small reward for taking a valid action, a medium reward for discovering new hosts for the first time, and a large reward for the goal.

These reward functions are designed to entice the agent into achieving the goal quickly without using any information the agent is not privy to, such as the agent's distance to the target.

Although this scenario is somewhat abstracted in comparison to a real-world network, it is representative of the tactical-level challenges an agent is faced with. When all 55 actions are available to a random agent at every step, then the agent has a 4 in 55^{15} (1 in $3.18e25$) chance of reaching the goal within 15 steps. Furthermore, each action will yield an effect and reward *at most* once, meaning an agent may collect disproportionately more samples of actions failing compared to samples of actions yielding an effect. In theory, this could complicate the agent's ability to learn actions that are only applicable late-game as the agent must first adequately learn the preceding steps before it can observe late-game samples.

A. Environment Mapping to Symbolic Logic

To stand up AR1ST0TLE for the CybORG environment, the set of predicates, object types, and formal definitions for each of the 5 above operators were derived from the CybORG code to form the PDDL domain definition. Additionally, a CybORG-to-PDDL translation layer was devised as part of the agent to convert the state into a PDDL problem file containing either the fully-observable state for classical planning, or the partially-observable state for AR1ST0TLE. This layer is also used to supply AR1ST0TLE with new observations directly at run-time and additionally forms the basis of the PPO's state space. The state of each individual host can be described by a boolean vector where each element indicates whether a predicate, such as `(superuser_access HOST)` or `(in_network HOST StartSubnet)`, has appeared in the observation at any point. It follows that the state can be described as a matrix where each column is a predicate and each row is a corresponding host.

V. Methods

A. Action Masking

Classical planners operate under the assumption that the world is fully-observable and deterministic, allowing the planner to reliably derive successor states of state-action pairs and select the best action based on some valuation of the resulting states. When faced with partial observability, however, there is no guarantee that the expected successor state adequately reflect reality: the only way to reliably generate an accurate successor state is to take an action in the environment and observe the changes. Consequently, when combined with the assumption that actions applied to a state are monotonic and irreversible, this prevents planners from backtracking to explore possible alternatives starting from a previous state.

In lieu of the certainty required to preview the next states and a utility function to rank them, it is possible to leverage a neural network to select the best action using only the current state and a candidate action. A PPO model is integrated with AR1ST0TLE to form a neuro-symbolic system that operates similar to an online planning system, where the symbolic component reduces the action space and the neural component selects the best action from the subset.

The PPO policy function is implemented as an NN that outputs a fixed-size vector ($\pi : \mathbb{R}^{|s|} \rightarrow \mathbb{R}^{|\mathcal{A}|}$) where each element of the vector corresponds to a specific action.The challenge arises when not all of the actions are available to the agent. At every step, only a subset of actions are valid actions that the agent can execute, either due to the action preconditions not being satisfied or the agent not knowing of the existence of the required parameters. Many such invalid actions exist in the action space of a red agent – for the presented experimental environment, over 80% of the 55 actions are invalid at any given time step. If there are no limitations to the actions available

to the agent, the agent must spend additional training time learning which actions are valid through trial-and-error.

The subset of applicable actions $\mathcal{A}_V$ is identified by AR1ST0TLE, and a mask $[a \in \mathcal{A}_V : a \in \mathcal{A}]$ is applied to the outputs corresponding to valid actions and the remaining values are set such that the agent cannot choose the invalid actions. Huang and Ontañón [15] proved that simply zeroing out elements corresponding invalid actions before applying the softmax function produces a valid policy gradient, and thus, a valid policy function π. Likewise, when choosing a random action, the sample weights of invalid actions are set to zero and the remaining weights are re-normalized accordingly. To prevent having to re-derive the action mask for every state when performing experience replay, the mask is conveniently stored alongside the state in the experience replay buffer.

B. AR1ST0TLE as an Environment Driver

The neuro-symbolic system requires an accurate description of the scenario environment and uses AR1ST0TLE to keep track of the agent's state during execution. As a consequence, AR1ST0TLE duplicates a substantial portion of CybORG's functionality needed to simulate the environment. In theory, using AR1ST0TLE in place of CybORG should accelerate training by eliminating any duplicated work at run-time. To fully replicate the functionality in practice, AR1ST0TLE only requires a reward function $\mathcal{R}$ and a transition function $\mathcal{T}$ to be specified in addition to the domain.

While AR1ST0TLE does not have the ability to represent numbers directly, preventing it from representing functions such as those based on the number of hosts under the agent's control, the rewards in Section IV are constrained to three discrete reward values. Accordingly, the reward is encoded as a conditional effect as a "reward" predicate, which a Gym interface wrapper around AR1ST0TLE then maps to a numeric value.

Similarly, the transition function $\mathcal{T}$ cannot be directly modelled in AR1ST0TLE, however, actions in experimental environment only have 2 possible outcomes: success, leading to a deterministic successor state, and failure, remaining in the current state. Valid actions have a fixed, non-zero probability of success, whereas invalid actions will always fail. As such, when the agent calls for a valid action, the wrapper will generate a random number in the range $[0, 1]$ and test whether the value falls within the action's success threshold. The wrapper will only forward actions to AR1ST0TLE when the action is deemed successful, returning an immediate failure to the agent and leaving the state unchanged otherwise.

VI. Experimental Results

A. Random Agent and Baseline PPO

A random agent without any constraints on the action space took an average of 722.62 steps ($\sigma = 180.56$), or $48.2\times$ the optimal plan length of 15, to reach the goal. As such, the cut-off for all agents was set just above this value at 750 steps per episode.

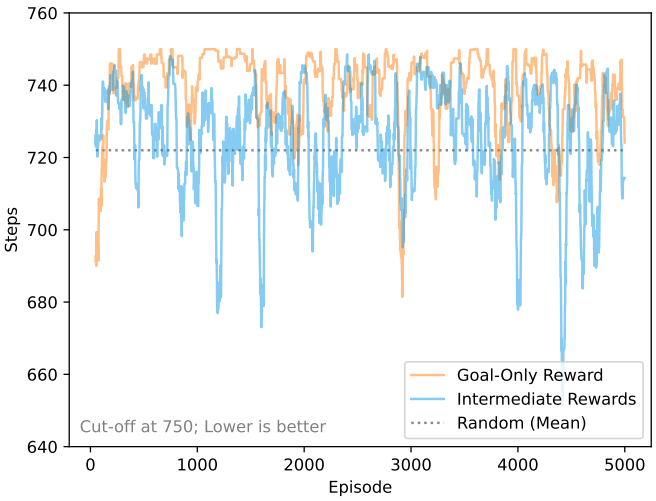

Fig. 2: Baseline PPO rolling average steps per episode.

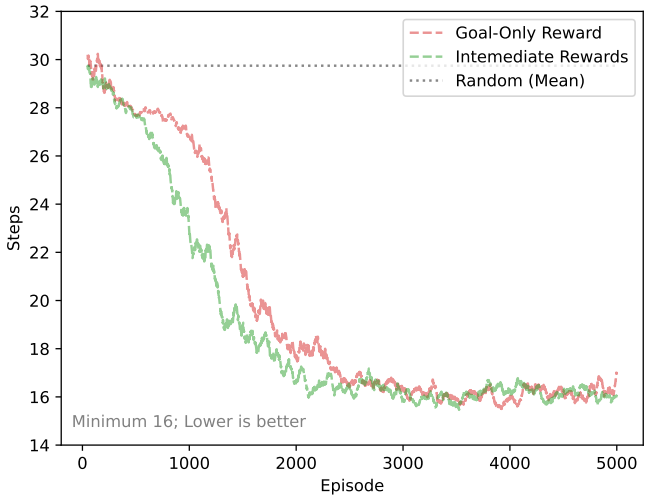

Fig. 3: Masked PPO rolling average steps per episode.

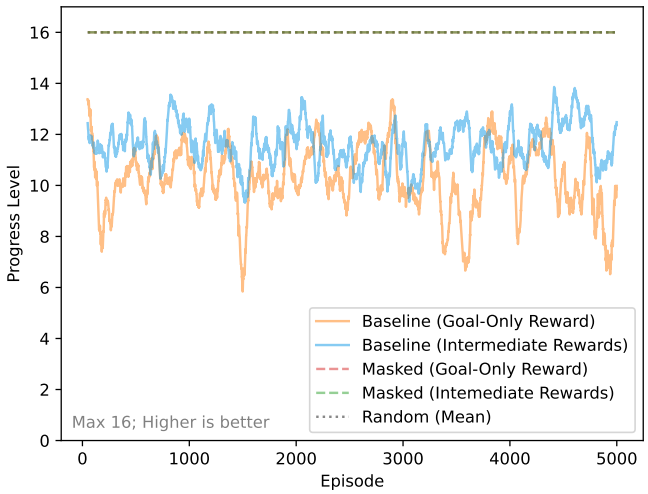

Fig. 4: Agent progress per episode. Each level corresponds to the plans outlined in Table I. Higher is better.

The Stable Baselines 3 PPO implementation was used as a baseline for this experiment [16]. As seen in figure 2, the baseline agents frequently achieved the goal of impacting the target server throughout training, however, these agents performed *worse* than the random agent on average. Figure 4 reveals that the agents were typically able to discover and exploit the last host in the enterprise subnet (step 9 and 10 in table I) on average. The agent with intermediate rewards managed an average of approximately 742 steps, on par with random, and the goal-only agent performed slightly worse at an average of approximately 738 steps.

B. Neuro-Symbolic Agent

This experiment used the Stable Baselines 3 Contrib Masked PPO implementation by Huang and Ontañón [15] using the same default hyper-parameters as the baseline. When compared to the random baseline, the random neuro-symbolic agent is capable of reaching the to just shy of 30 steps – only twice the optimal path length. This is a $\sim 96\%$ reduction in the average number of random steps required to reach the goal and represents a significant reduction in the problem complexity for the red agent. This is confirmed by the results in figure **??** where the agents learn the optimal path and complete episodes with an average of 16.3 ± 0.1 steps after 60k total steps. It is likely possible to further refine the results for both the masked agent and baselines through hyper-parameter turning, however, these results show that action masking is highly effective for achieving near-optimal results without requiring any additional tuning.

C. Environment Driver

To test whether using AR1ST0TLE as an environment driver is feasible, both CybORG and AR1ST0TLE were run in parallel to ensure the two environments were equivalent. Once this equivalency was verified, both environments were run for 100,000 steps with an agent that repeatedly executed a plan and an agent that selected actions at random. In both cases, the environment was reset when the agent reached the goal.

		Step Time	Reset Time	Total Time
Plan	CybORG	1.41ms	21.69ms	4m45s
	AR1ST0TLE	0.66ms	0.13ms	1m07s
Random	CybORG	1.38ms	21.64ms	2m20s
	AR1ST0TLE	0.02ms	0.13ms	3s

TABLE II: Average performance comparison over 100,000 steps between CybORG and AR1ST0TLE as an environment driver on a Ryzen Threadripper PRO 3995WX CPU running Ubuntu 20.04.

Table II provides a summary of the average step times, average reset times, runtimes from each trial. AR1ST0TLE outperformed CybORG by a noticeable margin, being $\sim 4.25\times$ faster in the plan trial and $\sim 47\times$ faster in the random trial.

The difference in AR1ST0TLE's performance between the plan and random trials is largely due to the fact that AR1ST0TLE identifies all valid actions in a single step and instantly fails any actions that did not appear in the set for the steps following. As a result of this "short-circuit," AR1ST0TLE only needs to perform computations when a valid action is executed. In contrast, CybORG's performance does not vary between trials as action validation is performed at every action execution.

Despite both environments having to perform this validation at every step for the plan-based agent, AR1ST0TLE was able to consistently outperform CybORG while managing to test the validity of all actions as opposed to an individual action. Moreover, AR1ST0TLE was nearly $167\times$ faster than CybORG in terms of resetting the environment, making up the remainder of the performance difference.

The plan trial is especially important as it reflects the expected performance for the neuro-symbolic agent described in Section VI-B where having the set of valid actions is vital for deciding the next action. When AR1ST0TLE is auxiliary to CybORG to generate the valid action space, the total runtime is closer to the individual runtimes combined (5m52s in this case). It follows that using AR1ST0TLE as an environment eliminates 81% of the overhead by removing CybORG from the equation, substantially accelerating RL training.

Given this acceleration, AR1ST0TLE was used in place of CybORG to train the RL agents and generate the results featured in the previous sections. The models were then validated on CybORG to ensure compatibility.

VII. Conclusion and Future Work

The CybORG scenario demonstrates the challenges of operating in a partially-observable environment. This research has shown that the intersection of RL and symbolic logic is highly effective for training agents quickly. A logic engine alleviates the need for an RL agent to learn the dynamics of an environment from scratch. Conversely, the use of the RL paradigm enables AI planning techniques in partially-observable domains without having to rigorously define rules and compute multiple plans. This research also introduced a novel use of a symbolic logic engine as light-weight environment driver, significantly speeding up agent training compared to the existing CybORG environment. Further research is required to evaluate whether this approach can be scaled to larger networks and environments with increasingly complex dynamics, and whether this approach generalizes to variations of the given scenario and different networks.

References

[1] C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis," in *Proceedings of the 1998 Workshop on New Security Paradigms*, ser. NSPW '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 71–79. [Online]. Available: https://doi.org/10.1145/310889.310919

[2] J. L. Obes, C. Sarraute, and G. Richarte, "Attack planning in the real world," in *Proceedings of the Second International Workshop on Security and Artificial*

Intelligence, AAAI 2010, vol. abs/1306.4044, 2013. [Online]. Available: http://arxiv.org/abs/1306.4044

[3] S. Jajodia, S. Noel, P. Kalapa, M. Albanese, and J. Williams, "Cauldron mission-centric cyber situational awareness with defense in depth," in *2011 - MILCOM 2011 Military Communications Conference*, 2011, pp. 1339–1344.

[4] D. Miller, R. Alford, A. Applebaum, H. Foster, C. Little, and B. E. Strom, "Automated adversary emulation: A case for planning and acting with unknowns," *MITRE Technical Papers*, 2018.

[5] J. Schwartz and H. Kurniawati, "Autonomous penetration testing using reinforcement learning," *CoRR*, vol. abs/1905.05965, 2019. [Online]. Available: http://arxiv.org/abs/1905.05965

[6] Microsoft Defender Research Team, "CyberBattleSim," https://github.com/microsoft/cyberbattlesim, 2021, 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.

[7] K. Tran, A. Akella, M. Standen, J. Kim, D. Bowman, T. Richer, and C. Lin, "Deep hierarchical reinforcement agents for automated penetration testing," in *Proceedings of the First International Workshop on Adaptive Cyber Defense, IJCAI 2021*, vol. abs/2109.06449, 2021. [Online]. Available: https://arxiv.org/abs/2109.06449

[8] M. Standen, M. Lucas, D. Bowman, T. J. Richer, J. Kim, and D. Marriott, "Cyborg: A gym for the development of autonomous cyber agents," in *Proceedings of the First International Workshop on Adaptive Cyber Defense, IJCAI 2021*, vol. abs/2108.09118, 2021. [Online]. Available: https://arxiv.org/abs/2108.09118

[9] L. Li, R. Fayad, and A. Taylor, "Cygil: A cyber gym for training autonomous agents over emulated network systems," in *Proceedings of the First International Workshop on Adaptive Cyber Defense, IJCAI 2021*, vol. abs/2109.03331, 2021. [Online]. Available: https://arxiv.org/abs/2109.03331

[10] A. Molina-Markham, C. Miniter, B. Powell, and A. Ridley, "Network environment design for autonomous cyberdefense," 2021.

[11] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

[12] B. Bonet and H. Geffner, "Planning under partial observability by classical replanning: Theory and experiments," in *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, T. Walsh, Ed. IJCAI/AAAI, 2011, pp. 1936–1941. [Online]. Available: https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-324

[13] M. Standen, M. Lucas, D. Bowman, T. J. Richer, J. Kim, and D. Marriott, "Cyborg: A gym for the development of autonomous cyber agents," *CoRR*, vol. abs/2108.09118, 2021. [Online]. Available: https://arxiv.org/abs/2108.09118

[14] J. Hoffmann, "Simulated Penetration Testing: From "Dijkstra" to "Turing Test++"," in *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*, R. I. Brafman, C. Domshlak, P. Haslum, and S. Zilberstein, Eds. AAAI Press, 2015, pp. 364–372. [Online]. Available: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10495

[15] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," *The International FLAIRS Conference Proceedings*, vol. 35, may 2022. [Online]. Available: https://doi.org/10.32473/flairs.v35i.130584

[16] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable Baselines," https://github.com/hill-a/stable-baselines, 2018.