RAMPART: Reinforcement Against Malicious Penetration by Adversaries in Realistic Topologies

Himanshu Neema, Daniel Balasubramanian, Harsh Vardhan, Harmon Nine, Sandeep Neema
Vanderbilt University, Nashville, TN, USA

***Abstract*—The increasing scale and complexity of networked computer systems, growing vulnerabilities and attack surfaces in the services and software running on these systems, and increasing reliance of mission critical workflows on such networked computer systems requires comprehensive cyber defense capabilities. However, manual monitoring and introspection often lacks the speed that these complex systems require for their defense. Therefore, it becomes crucial to design, develop, and train autonomous cybersecurity agents that can work with humans in defending the networked computer systems while sustaining the mission critical operational workflows. In this paper, we present a comprehensive framework for reinforcement learning (RL)-based cyber agent training called Reinforcement Against Malicious Penetration by Adversaries in Realistic Topologies (RAMPART). Using a model-driven architecture, RAMPART enables cybersecurity researchers to rapidly synthesize diverse scenarios, configurations and topologies (of networks, services, CVE-s, ports, workflows) on-demand, in a correct-by-construction manner, and with different levels of fidelity and abstraction, ranging from coarse-grained simulation, to hybrid simulation/emulation, to live networks. We envision that the configurability of the training environment and its instantiations at different levels of network simulation fidelity will allow RL approaches to scale and manage the complexity of the high-dimensional observation and action spaces.**

***Index Terms*—cyber-agent training, cybersecurity, reinforcement learning, network simulation, network emulation**

I. Introduction

Networked computer systems of today are continually increasing in scale and complexity due to increased networked connectivity, integration of distributed and cloud computing resources, and use of networked embedded control devices at the edge of the controlled systems and equipment [1]. The interdependencies of the system components and the complex interconnectivity through which they communicate and interoperate has significantly extended the attack surfaces. Furthermore, there has been an increasing reliance of mission critical workflows that run on top of networked computer systems. Therefore, it is crucial to build and automate comprehensive cyber defense capabilities for these system, while sustaining the mission critical operational workflows.

Traditional network administration and cybersecurity is largely based on network monitoring systems, networking rules, malware detection, and human interventions to safeguard, mitigate, and restore the system network. The challenge here is that the speed at which new vulnerabilities are being detected and cyber-attacks are carried out necessitates these methods to be automated in order to assist human cybersecurity experts to accelerate and optimize cyber defense actions. However, the task of developing such agents is daunting at multiple dimensions: (1) the sheer complexity and volume of data and information being processed by such networks; (2) diffusion of the context (e.g., the operational workflows) under which information and data is being processed across a myriad of software layers and network stacks; (3) the scale and diversity of actions that can be taken by agents across different layers of the networked environment (e.g., tuning firewall rules, configuring software services, deploying additional sensors, modulating workflow deployments and configurations); and (4) poorly understood effectiveness and side effects of actions within a highly complex network environment.

Reinforcement learning (RL) can potentially develop effective agents even within such challenging environments (as evidenced by successes of RL in many domains), but a key limiting factor is the lack of realistic RL training environments that can effectively model networked computer systems at scale, providing both the scale to perform millions of training epochs necessary for training RL agents, as well as a high-degree of realism so that trained agents' actions are representative and effective when deployed in real networks [2] [3].

In this paper, we present a comprehensive open-source framework (URL: https://github.com/CASTLEGym/) for reinforcement learning (RL)-based cyber agent training called the Reinforcement Against Malicious Penetration by Adversaries in Realistic Topologies (RAMPART). Using a model-driven architecture, RAMPART enables cybersecurity researchers rapidly sythesize diverse scenarios, configurations and topologies (of networks, services, CVE-s, ports, workflows) on-demand, in a correct-by-construction manner, and with different levels of fidelity and abstraction, ranging from coarse grain simulation, to hybrids of simulation and emulation, to live networks, all while preserving semantic correctness and realism of the environment including operational workflows. The framework is part of the ongoing research work and some of its functionality is not fully implemented but is envisioned.

RAMPART's RL training environment for network operations allows training both cyber-defense agents that learn defensive actions to maintain operational workflows, and cyber-attack agents that search for attack paths that exploit exposed network vulnerabilities. The three phases of the cyber agent training environment shown in Figure 1 are: (1) *Pre-training*: network and workflow configuration and scenario design; (2) *Training & Evaluation*: execution of training epochs through deployment, interspersed with validation and evaluation; and (3) *Post-training*: data collection, curation, and analyses.

979-8-3503-7594-7/24/$31.00 ©2024 IEEE
DOI 10.1109/DESTION62938.2024.00013

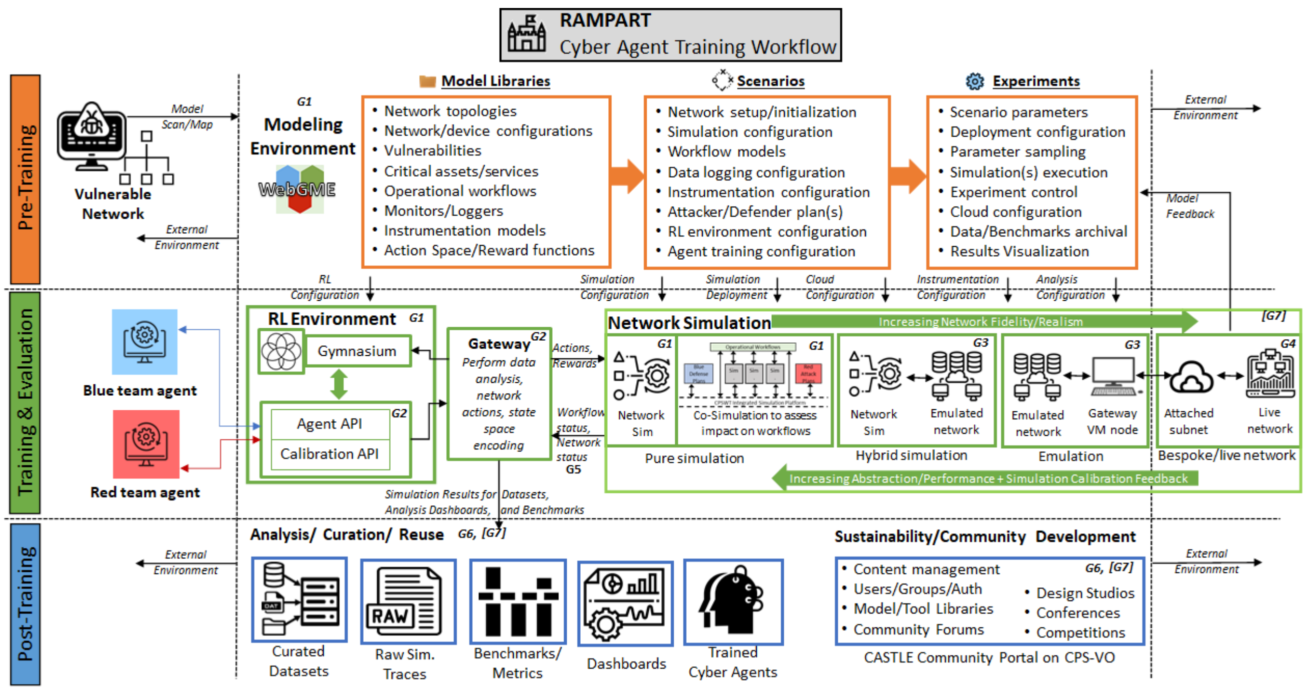


Fig. 1: RAMPART Technical Architecture.

In the pre-training phase, we use WebGME [4] modeling environment to design scenarios for training and experimentation. The integrated code generator in the environment generates configurations that are directly executable at different fidelity levels, viz. pure network simulation, emulation, and live networks.

In the training & evaluation phase, the configured system can be used for training agents, with different algorithms, in the RL environment. The agents are provided with observations (i.e., network state and operational workflow state), reward, and action APIs to train across configurable scenarios with different attack paths and mitigations. A key feature of the training environment is the generation of training datasets to enable offline cybersecurity algorithms development.

In the post-training phase, a detailed record of API actions and behaviors as well as data collected from deployed instrumentation are ingested into a cyber training database. A dashboard for data analytics over the recorded datasets is currently in development. In addition, the trained agents and the corresponding training environment APIs are also stored. This enables model feedback from human experts to further refine the training environment.

We envision that the configurability of the cyber agent training environment and multiple instantiations of different scenarios with fidelity and abstractions appropriate to various cyber agent behaviors and objectives will allow the RL approaches to scale and manage the complexity of the high-dimensional observation and action spaces.

II. Related Work

Noteworthy projects for training agents in cyber operations include Microsoft Research's CyberBattleSim [5], CybORG [6], [7] and CyGIL [8]. All provide an interface based on the ubiquitous OpenAI Gym API [9]. CyberBattleSim and CyGIL represent opposite design points in this space: simulation vs. emulation, trading off training speed for fidelity. CybORG provides both simulation and emulation modes and enables the transfer of trained agents to its emulator for optimization and validation. Preliminary results are encouraging, with a 66% successful transfer rate between simulation and emulation [7], albeit in a limited scenario consisting of several hosts in a flat network topology.

While the RAMPART architecture draws inspiration from these existing projects (e.g., OpenAPI Gym conformance, the use of simulation and emulation, the inclusion of actions drawn from standard taxonomies such as the MITRE ATT&ACK framework), it also differs in several key aspects to close the realism gap and enable scenarios that are closer in scale and depth to the complexity of real, bespoke networks. Distinguishing aspects of RAMPART include (1) using modeling and synthesis to instantiate gym environments, (2) providing a rich and flexible co-simulation framework, (3) scalable emulation using cloud computing, and (4) melding emulation with real-live network traffic. Furthermore, unlike other approaches that treat actions equally (e.g. all actions in CybORG take exactly one round to be reflected in the observable state), RAMPART supports the notion of delayed observables to model the fact that different actions will take different amounts of time to take effect in the real-world on a per bespoke network basis.

Adversary emulation tools like MITRE Caldera and Cobalt Strike enable modeling post-breach behavior of an attacker by scripting a series of commands to run remotely. We have extensive experience with using these tools to emulate fixed adversary campaigns, but they are not appropriate for simulation or for automatically emulating a reactive adversary.

The NSF has supported several projects through programs that provide community-wide resources and testbeds for networking research in realistic settings, such as Chameleon Cloud [10] and CloudLab [11] for cloud/edge computing, COLOSSEUM [12] and POWDER [13] for wireless networking, FABRIC [14] for at-scale research in networking and distributed systems, RENEW [15] for a programmable MIMO platform, AERPAW [16] for experimental aerial wireless networking, ARA [17] for smart and connected rural communities, and COSMOS [18] for city-scale wireless and mobile experimentation. These testbeds are valuable resources, but users must familiarize themselves with each one separately, and federating them is a difficult problem. Our proposed model-based design and automated orchestration and deployment capability will be able to set up a federation on the fly and at scale for cybersecurity researchers to evaluate their strategies.

III. Integration Architecture

Figure 2 shows RAMPART's agent evaluation architecture and the sections below briefly describe the key components of this architecture.

A. Modeling Environment

A WebGME environment customized with (1) an integrated set of domain-specific modeling languages (DSMLs) to specify different aspects of the training environment (e.g., network configuration and operational workflows); (2) code generators for generating co-simulation/emulation/live network deployments, the Gym environment, and the implementation of the blue/red agent APIs; and (3) a collection of model libraries and templates for commonly used topologies, services, workflows, action/observation APIs, and scenarios. Figure 3 shows the topology and scenario design aspects of RAMPART's modeling environment. As shown, here researchers can design custom network topologies as well as configure cyber agent parameters. The language is highly flexible and can be easily customized for future extensions.

B. Emulation Testbed for Agent Evaluation

For network emulation, we deployed the CAGE-2 network scenario [19] on an OpenStack cloud cluster. OpenStack is an open-source cloud computing platform that enables to manage and automate pools of compute, storage, and networking resources allowing users to deploy virtual machines and other resources to handle different workloads, and is highly scalable and flexible. OpenStack's API-driven design facilitates integration with existing systems and third-party applications, enhancing its capability to support a wide range of cloud services and applications.

Our network is defined using HEAT template, that describes the network infrastructure in a readable and declarative format. The HEAT template is written in YAML or JSON format. The template defines the resources that need to be created, such as virtual machines, networks, or storage, and specifies their properties and interdependencies. This is used by the HEAT orchestration engine to automate the deployment, scaling, and management of cloud resources. Specifically, for our emulation purposes, we selected the CAGE2 scenario, for emulating a real-world network configurations using OpenStack (see Figure 4). The CAGE2 scenario is part of the TTCP CAGE Challenge 2 [19], which focuses on the development of autonomous cyber defense agents. It involves a network divided into three subnets, with the goal of defending a critical operational server against a red agent that starts with access to a user machine in Subnet 1. The blue agent must use high-level actions to analyze hosts, remove malicious software, restore systems, and create decoy services to protect the network. The scenario is designed to develop strategies for responding to attackers across an entire network.

C. Velociraptor Interface

Velociraptor is an open-source tool utilized for endpoint monitoring, wherein the client and server communicate through a persistent connection. It employs a query language known as VQL (Velociraptor Query Language), which enables users to swiftly execute actions on the target host. These actions may include forensic investigations or custom queries. Predefined VQL queries, referred to as artifacts, are available for specific use cases. These artifacts can be easily shared and customized, facilitating the rapid deployment of new execution response capabilities from a centralized node to endpoints. In our experimental setting, Velociraptor is employed to execute custom actions defined in the CybORG for the blue agent. Conversely, the red agent, which is considered an adversary in this context, does not have access to Velociraptor and instead utilizes existing toolkits such as nmap, exploits, and shell commands to create connections and execute actions to attacked nodes.

D. Action Space for Cyber Agents

In the current phase of our work, we have selected a limited set of actions for both the red and blue agents. For the red agent, we have chosen the following actions: DiscoverRemoteSystems, DiscoverNetworkServices, ExploitRemoteServices, PrivilegeEscalate, and Sleep. For the blue agent, we have selected the following actions: Remove, Restore, Analyze, DecoyServices, and Sleep. The game setting in CAGE2 begins with an initial state in which the red agent has already compromised one of the hosts in the User subnet and used that compromised host to attack other hosts in the network. The compromised host in this case is 'User0,' which is a normal host with no confidentiality and availability factor. The red agent, having compromised User0, has SYSTEM-level access and can perform any desired actions on User0. These are the list of following actions that is executed in Emulation:

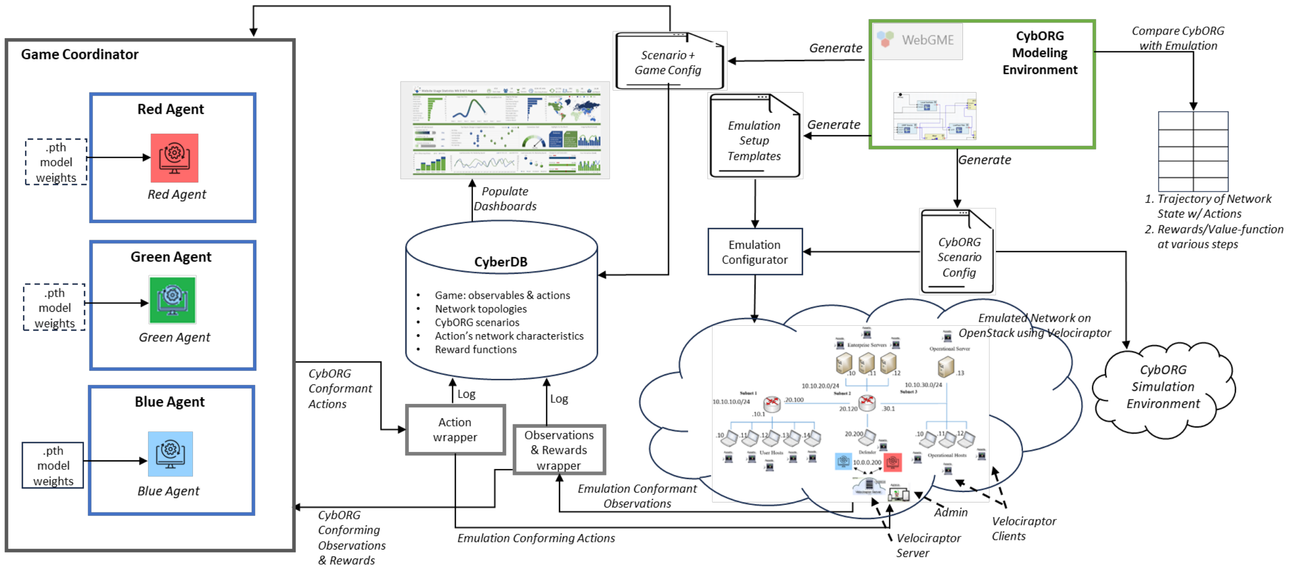
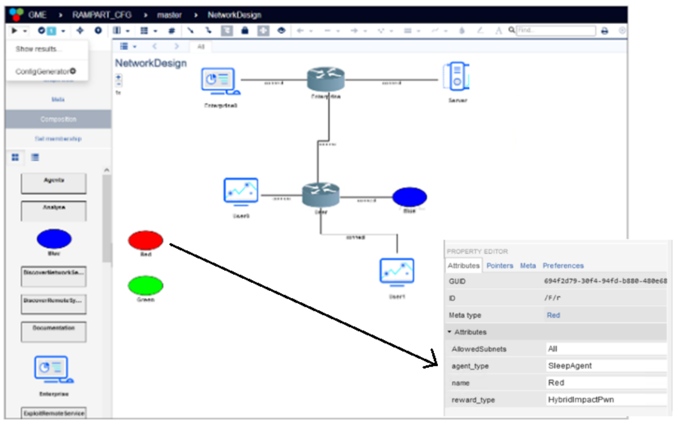

Game Coordinator
Red Agent
.pth model weights
Red Agent
Green Agent
.pth model weights
Green Agent
Blue Agent
.pth model weights
Blue Agent
Scenario + Game Config
Generate
WebGME
CybORG Modeling Environment
Compare CybORG with Emulation
Emulation Setup Templates
Generate
Generate
Populate Dashboards
CyberDB
Game: observables & actions
Network topologies
CybORG scenarios
Action's network characteristics
Reward functions
Emulation Configurator
CybORG Scenario Config
1. Trajectory of Network State w/ Actions
2. Rewards/Value-function at various steps
CybORG Conformant Actions
Log
Log
Action wrapper
Observations & Rewards wrapper
Emulated Network on OpenStack using Velociraptor
CybORG Simulation Environment
Emulation Conformant Observations
CybORG Conforming Observations & Rewards
Emulation Conforming Actions
Admin
Velociraptor Server
Velociraptor Clients

Fig. 2: Agent Evaluation Architecture.



GUID	694f2d79-30f4-94fd-b880-480e68
ID	/F/r
Meta type	Red
▾ Attributes	
AllowedSubnets	All
agent_type	SleepAgent
name	Red
reward_type	HybridImpactPwn

Fig. 3: WebGME Modeling Environment.

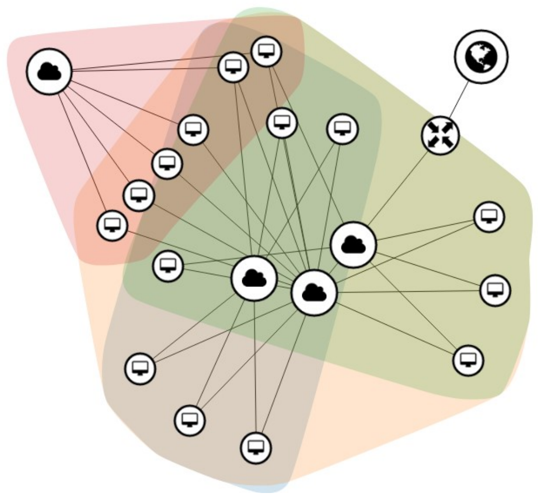


Fig. 4: CAGE2 Network Graph in OpenStack Cloud.

1) *DiscoverRemoteSystems*: This action is used to understand the network topology and explore other undiscovered hosts on the network. The input to this action is the IP address of hosts in CIDR notation that is in the network. The action is executed using the *nmap* command, with the assumption that *nmap* has been installed on the User0 host. The output of *nmap* is used to identify active hosts and their open ports.
2) *DiscoverNetworkServices*: This action also uses *nmap* to identify the network services running on the discovered hosts. The output provides information about the services and their associated ports.
3) *ExploitRemoteServices*: This action uses the network connection status to select exploits. If the exploit targets a decoy service, the exploitation fails. Otherwise, if it attempts to exploit an actual service, the exploitation succeeds.
4) *PrivilegeEscalate*: This action involves elevating the privileges of the red agent on the compromised host to gain greater control and access to resources and also explore the attacked host to gain any information. Currently it only seeks for IP address of unknown host who has interface level connection with the attacked host.
5) *Remove*: The blue agent uses this action to remove malicious files or processes identified on the host.
6) *Analyze*: The blue agent uses this action to analyze the network for signs of compromise or suspicious activity. It does it by two processes - first it runs *density-scout* on the host files and second it runs *signature-check* on host files. Any file with high entropy or failed signature check tells about the malware.
7) *DecoyServices*: This action involves setting up decoy services by the blue agent to mislead the red agent and protect real services. Decoys are like honey trap and if

red action tries to use it to run any exploits, its activity would be logged to blue agent and the exploit will fail.

8) *Sleep*: Both red and blue agents can use this action to do nothing for that step of execution, simulating a period of inactivity or waiting.

In the CAGE2 scenario, the actions ”Impact” and ”Monitor” are part of the available action set but are not included in the current emulation test bed.

E. Game Coordinator

The game coordinator python script is responsible for orchestrating the entire attack-defense game. The script is designed to work with both simulation and emulation environments, the environment selection can be changed by changing a single variable. Depending on the experiment type (sim for simulation or emu for emulation), the script creates a CybORG instance with the specified scenario and agents. For simulation, it uses the ‘B_lineAgent’ as the red agent, other agents can also be integrated by importing and changing the name. For emulation, it uses the ‘vu_emu’ class to create an emulation environment and sets up the blue and red agents. For blue agent a standard interface/wrapper called class ‘model_loader’ that provides a standardized interface for interacting with the agent and is responsible for loading and interacting with trained blue cyber agent. It abstracts away the details of the agent’s implementation, allowing for easy integration with different simulation scenarios and environments.

F. Emulation Wrapper

The initial state of emulation at the start of game is taken by initializing CybORG and modified according to network parameter in emulation to provide initial data structure that is meaningful to the agents. Both intial red and blue actions are transformed to suit the intial state of game and agents. he BlueEmulationWrapper class script serves as a wrapper for the blue team’s actions. It manages the blue team’s baseline information, current state observations, and the alter observation on the last action taken. The class provides functionality to reset the blue team’s state based on initial observations, update the state based on actions taken, and generate a table representing the current state. This table representation is fed into the blue agent for training as well as prediction. Similarlly red emulation wrapper transform the outputs from the red action to the CybORG compatible observation. Since blue montior the red action, the blue observations is further modified based on the red action and the red observation.

G. Agent Integration

The trained agent is loaded using the ‘model_loader’ class and this class is also responsible for interacting the agent during run time. There are following requirement that is imposed on agent class:

1) *Action Generation* (get_action method): This method should be part of agent class and its instance and it takes two arguments: observation object, which is the current state of the environment as perceived by the agent, and action_space, which defines the set of possible actions the agent can take.
2) *Episode Termination* (end_episode method): This method is called at the end of an episode to perform any necessary cleanup or reset operations. It calls the end_episode method of the agent instance.

H. Assessment on OT Networks

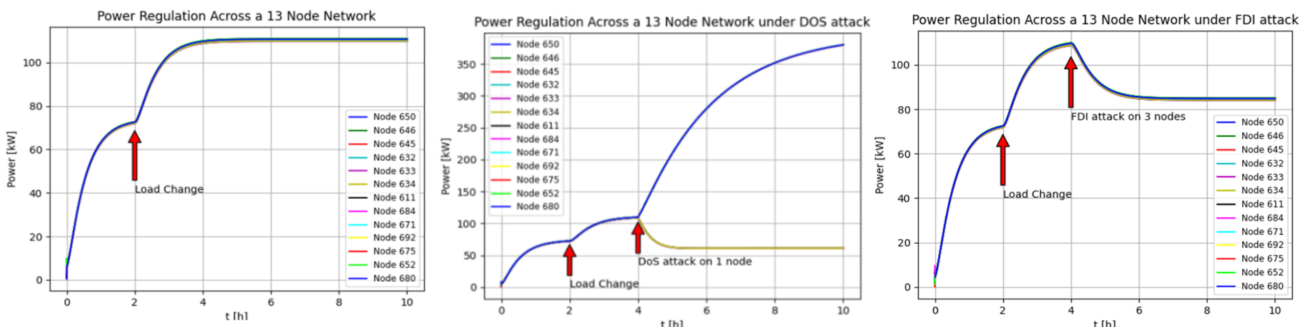
For Operational Technology (OT) deployment, our use case is distribution network of power grid network that is simulated by power grid simulation tool called Gridlab-D, and integrated with Python scripts. The python script can used to command and control the grid elements and its components and accordingly can translate the red and blue agent’s action in the grid network. Current implementation of OT service is a IEEE 13 bus feeder power network that is controlled by the control interface deployed at one of the host computer. We extend the grid model with a distributed consensus algorithm where local component-level controllers communicate the node-level state information to their neighbors until a consensus is achieved within the acceptable tolerance. In this experiment, we compared the baseline grid simulation with two cyber scenarios involving Denial-of-Service (DOS) (on node 8 at time 4 hours) and False Data Injection (FDI) (modifying reference voltage value from nodes 3, 4, and 5 to zero) attacks. Figure 5 compares the impact of these attacks against the baseline scenario.

IV. Lessons Learned

As we integrated the network emulation testbed with the CybORG’s RL-based cyber agent training environment, we noticed several limitations in CybORG as well as challenges with integrating it with network emulation.

A. Observed Limitations in CybORG

1) CybORG currently does not support the deployment or execution of actions, especially for the red agent. The intention of using the Velociraptor server-client-based endpoint monitoring is reflected in the Velociraptor agent class. However, the red agent is not allowed to access Velociraptor endpoints, and the means to deploy actions for the red agent is either completely ignored or standardized.
2) At each step, the blue agent observes the outcome of the action taken by the red agent and then takes an action.
3) After the DiscoverRemoteServices action, the blue agent observes that it is being scanned but does not take the most obvious action, such as blocking or isolating any further action or connection from the compromised node.
4) The most basic defense mechanism, like isolating the compromised node, is not available.
5) There is no file system; files are only written by the action of the red agent.
6) The Remove action automatically finds suspicious processes and removes them. However, there is no mechanism for the detection of suspicious processes, which is a difficult task in real-world scenarios.



(a) Power control across IEEE 13 node network using Distributed Controller – showing response to load changes.

(b) Power control across IEEE 13 node network under DDoS attack

(c) Power control across IEEE 13 node network under FDI attacks

Fig. 5: Evaluating Attacks on OT Networks.

7) The Detection and Isolation action, which uses density-scout, deems any file with a density of more than 0.9 as malicious. There are high chances of false positives for SSH keys, etc. Running density-scout and a signature-check on the entire file system is not feasible.
8) There is a connection between two subnet without the router in between it. And explorehost produces the IP address at the intefaces connected to the Userhost

B. Challenges in Emulation Integration

1) Creation of VMs with vulnerabilities: creating virtual machines (VMs) with specific software vulnerabilities requires the installation of versions of software containing those vulnerabilities. Configuring VMs in such a manner can be challenging due to the need for precise control over the software environment, which requires detailed knowledge of software dependencies and configurations. Ensuring compatibility between the software and the underlying VM infrastructure can be complex, especially when dealing with older versions that may not be supported by current systems. Additionally, sourcing the exact versions of vulnerable software can be difficult, as vendors often remove outdated versions from their official repositories to encourage users to update to more secure releases. Moreover, the process of configuring these VMs to replicate real-world scenarios while maintaining isolation from production environments for security purposes adds another layer of complexity to the setup.
2) Concrete implementation of abstract actions of cyborg in emulation: Translating abstract actions into concrete implementations that accurately reflect real-world behavior is a complex process. It requires understanding of how these actions manifest in actual network and there can be multiple implementation of an abstract action. Every implementation has dependencies as well as side effects that needs to be considered and managed.

V. Conclusion & Future Work

The ability to effectively train RL cyber agents depends crucially on a training environment that is scalable while simultaneously offering the realism necessary to successfully transfer the trained agents to real-world networks. In this paper, we presented a novel model-based cyber agent training environment that enables rapid synthesis of diverse scenarios with configurable network topologies, vulnerabilities, agent behavior, and deployment using abstract network simulation as well as emulation. The solution presented can significantly increase realism of the cyber agent training environments and help with developing autonomous agents that can work toward defending real-world networks.

In the future, we are working on increasing the available action and observation space for the agents, extending the modeling environment for further automating the deployment and configuration of network simulation and emulation, designing mechanisms to continually evolve the training environment for newer defense and attack methods, building integrated dashboards for agent evaluation and cyber games, developing methods to take lessons learned in higher-fidelity simulation to lower levels, and providing tools for incorporating direct feedback from humans for both agent training and evaluation.

VI. Acknowledgements

This work is supported by DARPA through contract number W912CG-23-C-0028. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and do not necessarily reflect the views of the sponsor.

References

[1] Dietmar PF Möller. Guide to cybersecurity in digital transformation.
[2] Thanh Thi Nguyen and Vijay Janapa Reddi. Deep reinforcement learning for cyber security. *IEEE Transactions on Neural Networks and Learning Systems*, 34(8):3779–3795, 2021.
[3] Amrin Maria Khan Adawadkar and Nilima Kulkarni. Cyber-security and reinforcement learning—a brief survey. *Engineering Applications of Artificial Intelligence*, 114:105116, 2022.
[4] Tamás Kecskés, Qishen Zhang, and Janos Sztipanovits. Bridging engineering and formal modeling: Webgme and formula integration. In *MODELS (Satellite Events)*, pages 280–285, 2017.
[5] Microsoft Research. CyberBattleSim. https://github.com/microsoft/CyberBattleSim/, 2020.
[6] Maxwell Standen, Martin Lucas, David Bowman, Toby J Richer, Junae Kim, and Damian Marriott. Cyborg: A gym for the development of autonomous cyber agents. *arXiv preprint arXiv:2108.09118*, 2021.
[7] Maxwell Standen, Martin Lucas, DavidBowman, Toby J.Richer, Junae Kim, and Damian Marriott. Cyborg: A gym for the development of autonomous cyber agents. *arXiv preprint arXiv:2108.09118v1*, 2021.

[8] Li Li, Raed Fayad, and Adrian Taylor. Cygil: A cyber gym for training autonomous agents over emulated network systems. *arXiv preprint arXiv:2109.03331*, 2021.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[10] Chameleon: A configurable experimental environment for large-scale edge to cloud research, March 2024.

[11] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, et al. The Design and Operation of CloudLab. In *USENIX Annual Technical Conference (ATC '19)*, pages 1–14, 2019.

[12] COLOSSEUM: The World's Most Powerful Wireless Network Emulator, March 2024.

[13] POWDER: Platform for Open Wireless Data-driven Experimental Research, March 2024.

[14] FABRIC: Adaptive Programmable Research Infrastructure for Computer Science and Science Applications, March 2024.

[15] RENEW: Reconfigurable Eco-system for Next-generation End-to-end Wireless, March 2024.

[16] AERPAW: Aerial Experimentation and Research Platform for Advanced Wireless, March 2024.

[17] ARA: Wireless Living Lab for Smart and Connected Rural Communities, March 2024.

[18] COSMOS: Cloud Enhanced Open Software Defined Mobile Wireless Testbed for City-Scale Deployment, March 2024.

[19] TTCP Cage Challenge 2, March 2024.