PenGym: Pentesting Training Framework for Reinforcement Learning Agents

Huynh Phuong Thanh Nguyen[1], Zhi Chen[1], Kento Hasegawa[2], Kazuhide Fukushima[2] and Razvan Beuran[1]

[1]*Japan Advanced Institute of Science and Technology, Japan*

[2]*KDDI Research, Inc., Japan*

Keywords: Penetration Testing, Reinforcement Learning, Agent Training Environment, Cyber Range.

Abstract: Penetration testing (pentesting) is an essential method for identifying and exploiting vulnerabilities in computer systems to improve their security. Recently, reinforcement learning (RL) has emerged as a promising approach for creating autonomous pentesting agents. However, the lack of realistic agent training environments has hindered the development of effective RL-based pentesting agents. To address this issue, we propose PenGym, a framework that provides real environments for training pentesting RL agents. PenGym makes available both network discovery and host-based exploitation actions to train, test, and validate RL agents in an emulated network environment. Our experiments demonstrate the feasibility of this approach, with the main advantage compared to typical simulation-based agent training being that PenGym is able to execute real pentesting actions in a real network environment, while providing a reasonable training time. Therefore, in PenGym there is no need to model actions using assumptions and probabilities, since actions are conducted in an actual network and their results are real too. Furthermore, our results show that RL agents trained with PenGym took fewer steps on average to reach the pentesting goal—7.72 steps in our experiments, compared to 11.95 steps for simulation-trained agents.

1 INTRODUCTION

Network security plays a critical role in our current network-centric society. Penetration testing (pentesting) is an important aspect of cybersecurity that involves assessing the security posture of networks or systems by conducting ethical cyberattacks on them. However, traditional pentesting has significant challenges to overcome, such as the lack of IT professionals with sufficient skills. Recently, there has been a growing interest in applying machine learning techniques to automate and improve pentesting.

In this context, reinforcement learning (RL) has emerged as a promising approach for training agents to perform pentesting tasks in a more effective manner. Thus, RL agents aim to replicate the actions of human pentesters, but with the speed, scale, and precision that only programs can achieve. This is achieved by making it possible for the RL agents to navigate complex network environments, detect vulnerabilities, and exploit them to evaluate security risks. Fundamentally, through a process of trial and error, the RL agents learn to optimize their actions by adapting to various environment challenges (Zhou et al., 2021).

However, so far RL agents have been trained and are performing penetration testing mainly in simulated environments. Simulators provide an in-memory abstraction of processes that occur in real computer networks, which makes them faster and easier to use than their real counterparts. However, simulators often suffer from a "reality gap", as the level of abstraction used in simulator makes it difficult to deploy the trained agents in real networks. For example, the authors of CyberBattleSim themselves argue that their framework is too simplistic to be used in the real world (Microsoft Defender Research Team, 2021). This means that agent performance may suffer when used with real networks due to the differences with the simulated environment. In particular, the translation of simulated actions (e.g., exploits, privilege escalation) to real actions is not trivial. As a result, creating and operating realistic environments for the training of pentesting AI agents is crucial.

To address this issue, we have developed PenGym, a framework for training RL agents for pentesting purposes using a real cyber range environment. The key feature of PenGym is that it makes it possible for agents to execute real actions in an actual network environment, which have real results that correspond to RL agent state and observations. Thus, PenGym elim-

Nguyen, T., Chen, Z., Hasegawa, K., Fukushima, K. and Beuran, R.
PenGym: Pentesting Training Framework for Reinforcement Learning Agents.
DOI: 10.5220/0012367300003648

In *Proceedings of the 10th International Conference on Information Systems Security and Privacy (ICISSP 2024)*, pages 498-509
ISBN: 978-989-758-683-5; ISSN: 2184-4356

inates the need to model agent actions via execution assumptions and success probability. Therefore, our approach provides a more accurate representation of the pentesting process, since everything is based on actual network behavior, and yields more realistic results than simulation.

The effectiveness of PenGym has been validated through several experiments, which demonstrated its reasonable training time and suitability as an alternative to simulation-based environments. Moreover, the trained agents were used successfully to conduct real pentesting attacks in the cyber range.

By using PenGym, security researchers and practitioners can train RL agents to perform pentesting tasks in a safe and controlled environment, thus obtaining more realistic results than via simulation, but without the risks associated with real network pentesting. By providing the environment for executing actions, the framework can also be used to evaluate and compare the effectiveness in real network environments of various pentesting RL techniques.

The main contributions of this paper are:

- Present the design and implementation of PenGym, with particular focus on the action implementation that represents its key feature.
- Discuss a set of experiments that demonstrate the potential of using PenGym to effectively train RL agents in pentesting when compared to simulation-based training.

The remainder of this paper is organized as follows. In Section 2, we discuss related research works. In Section 3, we provide an overview of the PenGym architecture, followed by a detailed description of the action/state module implementation in Section 4. We then present the results of the validation experiments in Section 5. The paper ends with a discussion, conclusions, and references.

2 RELATED WORK

The field of cybersecurity has seen a growing use of cyber range network environments as a popular training method for cybersecurity professionals. Moreover, recent studies have explored the design and implementation of cyber range environments for conducting cyber attack simulations and for training RL agents in tasks such as intrusion detection, malware analysis, and penetration testing. We will discuss some of the most representative studies below; a summary of their characteristics when compared to PenGym is given in Table 1. All approaches are compared based on their abstraction level and execution environment features. Regarding the abstraction level, simulation-based approaches use a simulation environment to execute actions. In these approaches, actions are modeled by checking several required conditions, and returning success if all the conditions are met (Schwartz and Kurniawati, 2019). On the other hand, emulation environments require actual hosts, an actual network topology, and agents that execute real actions on those hosts (Li et al., 2022). When considering execution environment features, the configurable elements are used for comparison, including features such as firewalls and host actions.

Several frameworks were developed for cyber range training. SmallWorld (Furfaro et al., 2018) and BRAWL (The MITRE Corporation, 2018) use cloud-based infrastructure and virtualization technologies to simulate user interaction with a host, but they lack RL capabilities. While some training environments for AI-assisted pentesting that focus on host-based exploitation have been proposed in previous studies (Pozdniakov et al., 2020)(Chaudhary et al., 2020), their scope of game goals and available actions is quite limited. In (Ghanem and Chen, 2018), the authors proposed a training environment for network penetration testers modeled as a Partially Observable Markov Decision Process (POMDP), but details of the environment and reinforcement learning training were not provided. Another experimental testbed for emulated RL training for network cyber operations is Cyber Gym for Training Autonomous Agents over Emulated Network Systems (CyGIL) (Li et al., 2022). CyGIL uses a stateless environment architecture and incorporates the MITRE ATT&CK framework to establish a high-fidelity training environment.

Network Attack Simulator (NASim) (Schwartz and Kurniawati, 2019) proposed an RL agent training approach for network-wide penetration tests using the API of OpenAI Gym (Brockman et al., 2016). NASim represents networks and cyber assets, including hosts, devices, subnets, firewalls, services, and applications, using abstractions modeled with a finite state machine. The simplified action space includes network and host discovery, service exploitation for each configured service vulnerability in the network, and privilege escalation for each hackable process running in the network. The agent can simulate a simplified kill chain through discovery, privilege escalation, and service exploits across the network. However, NASim assumes that the simulated actions must satisfy various predefined conditions, and uses probabilities to determine their success.

Microsoft has recently open-sourced its RL agent network training environment, the CyberBattleSim (CBS) (Microsoft Defender Research Team, 2021),

Table 1: Comparison of related frameworks from abstraction level and environment feature perspectives.

	Small World	BRAWL	Smart Security Audit	CyGIL	Microsoft CBS	CybORG	FARLAND	NASim	PenGym (Ours)
Abstraction Level									
Simulation Based					✓	✓	✓	✓	✓
Real Hosts	✓	✓	✓	✓		✓	✓		✓
Real Network Topology	✓	✓		✓		✓	✓		✓
Real Actions	✓	✓	✓	✓		✓	✓		✓
Real Observations	✓	✓	✓	✓		✓	✓		✓
Designed for RL			✓	✓	✓	✓	✓	✓	✓
Host-Based Exploitation	✓	✓	✓	✓	✓	✓		✓	✓
Network-Based Exploitation	✓	✓		✓			✓	✓	✓
Environment Features									
Firewalls	✓							✓	✓
Network Scanning	✓	✓		✓			✓	✓	✓
Host Scanning (OS Scan, Process Scan, Service Scan)			✓					✓	✓
Exploits	✓	✓	✓	✓	✓	✓		✓	✓
Privilege Escalation	✓		✓	✓	✓	✓		✓	✓

which is also built using the OpenAI Gym API. CBS is designed for red agent training that focuses on the lateral movement phase of a cyberattack in a simulated fixed network with configured vulnerabilities. Similar to NASim, CBS allows users to define the network layout and the list of vulnerabilities with their associated nodes. It is important to note that CBS is stated to have a highly abstract nature that cannot be directly applied to real-world systems.

CybORG (Standen et al., 2021) is a gym for training autonomous agents through simulating and emulating different environments using a common interface. It supports red and blue agents, and implements different scenarios at varying levels of fidelity. Actuator objects facilitate interactions with security tools and systems, such as executing real actions through APIs or terminal commands, using the Metasploit pentesting framework (Maynor, 2011). The focus of CybORG lies in developing an autonomous pentesting agent using RL and host-based exploitation, although it does not consider network traffic discovery or connections between subnets.

The FARLAND framework (Molina-Markham et al., 2021) is designed for training agents via simulation and testing agents via emulation. It offers functionality such as probabilistic state representations and support for adversarial red agents. However, unlike CybORG, FARLAND focuses on network-based discovery instead of host-based exploitation.

To summarize, all of the frameworks mentioned above have their own limitations. Thus, some of them are designed to support real environments with only a few actions, such as exploits and privilege escalation (Furfaro et al., 2018) (The MITRE Corporation, 2018) (Li et al., 2022). Moreover, some of those systems fail to consider the host configuration before attempting an attack, and are not designed for RL purposes(Furfaro et al., 2018) (The MITRE Corporation, 2018). Other approaches focus on either host-based exploit actions (Microsoft Defender Research Team, 2021) (Standen et al., 2021), or network-based actions (Molina-Markham et al., 2021), but not both of them. Only NASim (Schwartz and Kurniawati, 2019) supports both host-based and network-based actions, including external firewalls, but it uses a simulation environment to carry out these actions.

Given the limitations of NASim, and taking inspiration from the emulation approaches discussed so far, we have developed PenGym as an extension of the NASim library that makes it possible to both train and use pentesting RL agents in real network environments. PenGym covers both network traffic discovery and host-based exploitation actions that are all actually conducted in the emulated environment.

3 PenGym OVERVIEW

PenGym is a framework for creating and managing real environments aimed at training RL agents for penetration testing purposes. It provides an environment where an RL agent can learn to interact with a network environment, carrying out various penetration testing tasks, such as exploit execution, and privilege escalation. PenGym uses the same API with the OpenAI Gymnasium (formerly Gym) library, thus making it possible to employ it with all the RL agents that follow those specifications.

An overview of PenGym is shown in Figure 1. First, the RL agent selects an action from the available space using an algorithm suited to its learning objectives. Subsequently, PenGym converts this logical action into an executable real action, which is then executed in the cyber range environment set up using KVM virtual machines. Following action execution, the actual observations, such as available services or exploit status, along with the new state of the environment, are interpreted by PenGym and sent back to

the agent. The agent receives a corresponding reward and relevant system information. The rewards are defined via a scenario file, which contains different positive rewards for each successfully executed action. In case of failure, the agent receives a negative reward. For the scope of this paper, the reward values used adhere to those predefined in NASim. The agent then updates its learning algorithm to select the next suitable action. For example, if an observation shows a certain available service, the agent can use an exploit based on that service for the next step. The use of the acquired state information and corresponding reward to refine and enhance the underlying algorithm, ultimately fosters the RL agent's learning and optimization process. This key functionality of PenGym is implemented via its core component, the *Action/State Module*, which has two main roles:

- Convert the actions generated by the RL agent into real actions that are executed in the *Cyber Range* environment.
- Interpret the outcome of the actions and returns the state of the environment and the reward to the agent, so that processing can continue.

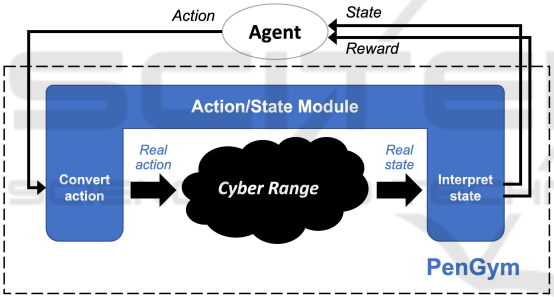

Figure 1: Overview of the PenGym architecture.

The *Action/State Module* module implements a range of penetration testing techniques that are executed on actual target hosts, thus enabling penetration testing in conditions similar to real-world scenarios.

The *Cyber Range* in PenGym is created using KVM virtualization technology to build a custom network environment that hosts virtual machines, with each cyber range consisting of a predefined group of hosts, services, and vulnerabilities that are made available for the interaction with the RL agent. The composition and content of a cyber range is determined based on the content of NASim scenario files in order to build an equivalent environment in PenGym.

NASim scenario files act as a blueprint for the cyber range environment and a guide for the behavior of the agents. In particular, they define: (i) the network environment, such as the characteristics of the various hosts (e.g., operating system, processes and services, etc.), subnets and firewall rules between hosts, and (ii) the actions that RL agents can take, how rewards are obtained and what pre-conditions are necessary for the agents to perform those actions successfully.

4 IMPLEMENTATION

In NASim, the action space is a collection of feasible actions an agent can execute. These actions includes various scan actions (e.g., service scan, OS scan, process scan, and subnet scan) that help identify vulnerabilities and access points in the network. They mimic the functionality of the Nmap utility (Lyon, 2014), providing information about active services and the operating system running on a specified host.

The action space in NASim allows for exploit actions on services and machines in the network, which can lead to unauthorized access and the exploitation of vulnerabilities. The success of an exploit action is determined by factors such as the existence of the target service, firewall rules, and the success probability. Privilege escalation, a tactic to gain higher access, is also included. Thus, NASim simulates various network security mechanisms.

One major limitation of using simulated actions as done in NASim is that those actions may not accurately replicate the real-world behavior. Although the result of an action is determined by checking specific conditions in the description file, it is important to recognize that other factors can also impact the success of the action. Moreover, the network environment itself may not be accurately replicated in simulation, including the configuration and topology of the network. Therefore, while simulation can provide valuable insights into network security, real-world execution is the only way to determine the effectiveness of penetration testing agents.

The *Action/State Module* in PenGym aims to bridge this gap by enabling the actual execution of RL agent actions in the target cyber range. The outcome of each action is determined based on the current status of the virtual machine (VM) host in the network environment, reflecting the real conditions of the system. The actions currently implemented in PenGym cover the entire functionality of NASim as required by the scope of the 'tiny' scenario used in our experiments, and can be extended for other scenarios. The PenGym action space is divided into six categories:

1. Service Scan
2. Operating System (OS) Scan
3. Subnet Scan
4. Exploit

5. Process Scan
6. Privilege Escalation

For Service Scan, OS Scan and Subnet Scan, the real action implementation leverages the Nmap utility (Lyon, 2014) to retrieve information about the services and operating system of each host, and the accessible subnets. More specifically, we use the `python-nmap` library (Norman, 2021) to control Nmap via the Python programming language.

The Exploit and Privilege Escalation actions leverage the Metasploit penetration testing framework (Maynor, 2011) to actually execute the corresponding actions on the real target hosts. In particular, we use the `pymetasploit3` library (McInerney, 2020) to control Metasploit execution via Python.

As for Process Scan, it is implemented by using the `ps` command from the Linux operating system.

Note that, in order to reduce the execution time of the actual actions, after the first successful execution of an action, the relevant information regarding the result of that action is stored in a host map dictionary and reused for subsequent executions. This dictionary is used only for a single pentesting execution period in testing. This period ends when the target hosts are compromised or the step limit is exceeded. Similarly, it is used for a single training time in training, which ends when all the training epochs are finished. At the beginning of each testing period or training time, the dictionary is reset to an empty state. This ensures that the optimization strategy does not affect the realism of the overall training process or the evaluation of trained agents during testing. It helps minimize time by avoiding redundant actions that have already been successful in the same period. This mimics real-world situations where a pentester does not repeat successful actions. Therefore, using the host map dictionary in PenGym helps optimize execution time while maintaining training realism. The following sections provide more details about each action implementation.

4.1 Service Scan

Service scanning is used to identify and provide details about the services that are running on a host. It can also aid in the detection of potential vulnerabilities in those services. To implement the Service Scan action, we make use of the Nmap utility, which makes it possible for PenGym to identify a wide range of services, including web servers, SSH services, etc. Upon success, the list of services running on the target host is returned. The pseudocode for the implementation is provided in Algorithm 1. Several arguments are used to minimize the execution time of Nmap (`-Pn` to disable `ping` use, `-n` to disable DNS resolution, and `-T5` to enable the most aggressive timing template), and `-sS` is used to enable TCP SYN scanning.

Algorithm 1: Service Scan Action.

```
Require: host, nmap, port=False
  if port exist then
     result ← nmap.scan(host,port,arguments
= 'Pn -n -sS -T5')
  else
     result ← nmap.scan(host,arguments='-Pn
-n -sS -T5')
  end if
  service_list ← list()
  for host in result do
     if host[port][state] is OPEN then
        service_list.append(service_name)
     end if
  end for
  Return service_list
```

4.2 OS Scan

OS scanning is used to identify the operating system that is running on a target host. This functionality works by sending a series of probes to the target machine and analyzing the responses to determine the characteristics of the operating system. The accuracy of the scan results depends on the response from the target machine and the effectiveness of the probing technique used. The OS Scan action is implemented via the Nmap utility. Upon success, the action will return the list of potential operating systems running on the target host as identified Nmap. The pseudocode for the implementation is provided in Algorithm 2. In addition to time-optimization arguments, the argument `-O` is used to activate OS detection.

Algorithm 2: OS Scan Action.

```
Require: host, nmap, port=False
  if port exist then
     result ← nmap.scan(host,port,arguments
= '-Pn -n -O -T5'
  else
     result ← nmap.scan(host,arguments='-Pn
-n -O -T5'
  end if
  os_list ← list()
  for item in result do
     os_list.append(get_os_type(item))
  end for
  Return os_list
```

4.3 Subnet Scan

Algorithm 3: Subnet Scan Action.

```
Require: subnet, nmap, port=False
  host_list ← list()
  if port exist then
     result ← nmap.scan(subnet,
  port, arguments = '-Pn -n -sS -T5
  -minparallel 100 -maxparallet 100')
     for host in result do
        if host['tcp'][port][state] is OPEN
  then
           host_list.append(host)
        end if
     end for
  else
     result ← nmap.scan(host, arguments
  = '-Pn -n -sS -T5 -minparallel 100
  -maxparallel 100')
     for host in result do
        if host[status][state] is UP then
           host_list.append(host)
        end if
     end for
  end if
  Return host_list
```

Subnet scanning is a type of scan used to identify the active hosts within a specified network range, so that the potential targets in that subnet can be determined. Nmap subnet scanning works by sending a `ping` message to each IP address within the specified network range and then analyzing the responses to determine which hosts are active.

The Subnet Scan action in PenGym is implemented by using Nmap to scan the specified network range and retrieve the active hosts. Upon success, the list of the discovered hosts is returned; note that we differentiate between already discovered and newly discovered hosts, so that new potential targets can be easily identified. The implementation pseudocode is provided in Algorithm 3. In addition to time optimization, `-minparallel` and `-maxparallel` are used to enable the parallel probing of the hosts.

4.4 Exploit

Exploits are techniques for finding and taking advantage of vulnerabilities in software or systems to gain unauthorized access or perform malicious actions. A successful exploit will result in the target machine becoming compromised, and further steps can be performed, such as stealing sensitive data, installing malware, or taking control of the system. Therefore, exploits are critical components of the penetration testing process for advancing towards a target.

In PenGym, the Exploit action is implemented via the Metasploit (Maynor, 2011) framework. Upon successful completion, the shell object that makes it possible to access the target host, and the access level (typically "USER") are returned. The returned shell object can be used to execute shell commands or navigate through the file system. The returned access level can be used to determine what actions are allowed or restricted for the current user. The pseudocode for the implementation is provided in Algorithm 4. The Exploit action is currently implemented via an SSH exploit based on the dictionary attack technique.

Algorithm 4: SSH Exploit Action.

```
Require: host
  msfprc ← get_msfprc_client()
  shell ← check_shell_exist()
  if shell exist then
     shell ← get_existed_shell_of_host()
     access_level ← get_host_access_level()
     Return shell, access_level
  else
     exploit_ssh ← msfrpc.modules.use
     ('auxiliary','auxiliary/scanner/ssh/
  ssh_login')
     exploit_ssh['rhost'] ← host
     exploit_ssh['username'] ← username
     exploit_ssh['pass_file'] ← pass_file
  end if
  result ← exploit_ssh.execute()
  shell ← get_shell(result)
  access_level ← get_host_access_level()
  Return shell, access_level
```

4.5 Process Scan

Process scanning enables pentesters to conduct a thorough security assessment by identifying the processes running on a target host. This is done in view of determining which vulnerabilities can potentially be exploited for getting further control of the host, such as via privilege escalation techniques. Note that process scanning requires access to the target host, hence it is executed after successfully gaining access to it.

The Process Scan implementation in PenGym utilizes the shell object obtained via the Exploit action, and uses it to execute the `ps` command, which collects information about the processes running on the target host; upon success, the list of processes is returned. To speed up process information extraction, the `-A` and `-o` options are used to reduce execution time by extracting only essential user values associated with

the running processes, minimizing unnecessary overhead. This approach ensures a faster retrieval of process information and simplifies subsequent management of the extracted data (the implementation pseudocode is not included due to space limitations).

4.6 Privilege Escalation

Algorithm 5: Privilege Escalation Action.

```
Require: host
  msfprc ← get_msfprc_client()
  root_shell ← check_root_shell_exist()
  normal_shell_id ← get_normal_shell_id()
  if root_shell exists then
     root_shell ← get_root_shell()
     access_level ← get_host_access_level()
     Return root_shell, access_level
  else
     exploit_pkexec ← msfrpc.modules.use
     ('linux/local/cve_2021_4034_pkexec')
     exploit_pkexec['session'] ← shell_id
     payload ← 'meterpreter/reverse_tcp'
  end if
  result ← exploit_pkexec.execute()
  root_shell ← get_shell(result)
  access_level ← get_host_access_level()
  Return root_shell, access_level
```

Privilege escalation is an essential step in pentesting by which one attempts to gain administrator (root) access on the target system. By obtaining a higher access level than that of a regular user, a pentester gains complete control of the target system and can perform any actions on it. Privilege escalation is achieved by exploiting specific vulnerabilities or misconfigurations of the system to gain root level access. Note that privilege escalation is conducted after successfully gaining regular user access to the target host.

The implementation of the Privilege Escalation action utilizes the shell object obtained via the Exploit action to execute a privilege escalation exploit. In particular, we use the CVE-2021-4034 vulnerability in the `pkexec` program (National Vulnerability Database, 2021), for which a corresponding module is implemented in Metasploit, named `linux/local/cve_2021_4034_pkexec`. The implementation pseudocode is shown in Algorithm 5.

Optimizing the execution time of privilege escalation is an important part of making PenGym run efficiently, and we achieved this as follows:

- Disabled the `AutoCheck` attribute in Metasploit.
- Return the result of the privilege escalation function as soon as a meterpreter is created, instead of waiting for the job to finish.
- Only clean the Metasploit sessions and jobs after an attack sequence ends.

5 EXPERIMENTS

In this section we discuss first the main experiment scenario, then the action implementation validation, followed by several RL agent experiments.

5.1 Main Experiment Scenario

We used the scenario named 'tiny' defined in NASim, which is illustrated in Figure 2, both for RL agent training and testing. This scenario consists of three hosts divided into three subnets. Subnet(1) is directly connected to the Internet, and the other subnets are internally connected. Each host has the same basic configuration, with a Linux OS, SSH service, and Tomcat process. Firewall rules are enforced for secure communication between subnets, and only SSH communication is allowed. Specifically, Subnet(1) is accessible from the Internet via SSH but cannot connect externally. SSH access is allowed between subnets, except for the connection from Subnet(1) to Subnet(2). In practice, these restrictions were implemented via the `iptables` Linux firewall configuration utility.

For experiment purposes, we used KVM technology to create a network environment that is based on the 'tiny' scenario. This environment included all the configurations defined in this scenario, such as the hosts and their settings, and the composition of the subnets. In our experiments specific techniques are used for exploits and privilege escalation. Namely, an SSH dictionary attack is utilized to login to hosts, and a `pkexec` vulnerability is used to obtain root access. User accounts and passwords were set up accordingly on the target hosts to enable these actions. The experiments were conducted on a dual 12-core 2.2 GHz Intel(R) Xeon(R) Silver 4214 CPU server with 64 GB RAM. The NASim version we utilized was v0.10.0.

5.2 Action Implementation Validation

Table 2 provides a summary of the action implementation in NASim and PenGym, with differences highlighted in bold font. We also show the action execution times (for PenGym only for the very first execution of an action within a session, since we cache the output data to speed up subsequent execution). Next we discuss the validation of action implementations in PenGym to demonstrate their equivalence with NASim actions in terms of observations.

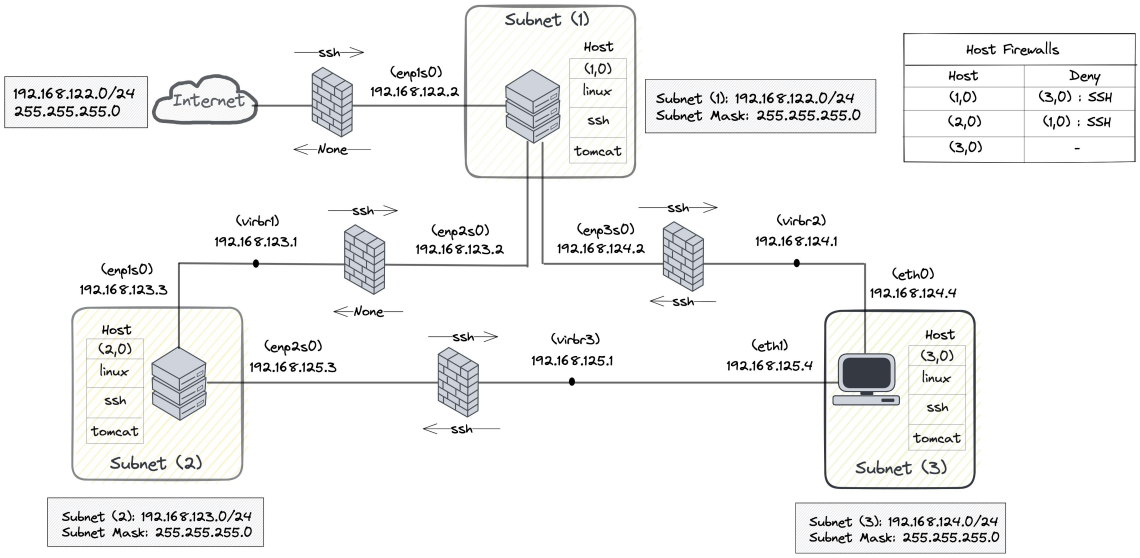

Figure 2: Cyber range constructed in PenGym based on the 'tiny' scenario in NASim.

Table 2: Comparison of action implementation and execution time between NASim and PenGym.

Action	NASim Implementation	Execution Time [s]	PenGym Implementation	Execution Time [s]
OS Scan	Check if the host is discovered and return its configured OS type	0.000006	Check if the host is discovered and **scan the OS type using `nmap`**	3.642790
Service Scan	Check if the host is discovered and return its configured services	0.000004	Check if the host is discovered and **scan the running services using `nmap`**	0.274271
Exploit	Check if the host is discovered and probabilistically update its state to "compromised"	0.000023	Check if the host is discovered and **execute an SSH dictionary attack to compromise it**	1.217554
Subnet Scan	Check if the host is compromised and return the connected hosts within the subnet	0.000028	Check if the host is compromised and **scan the connected hosts within its subnet using `nmap`**	2.486091
Process Scan	Check if the host is compromised and return its configured processes	0.000005	Check if the host is compromised and **scan the running processes using `ps`**	0.256476
Privilege Escalation	Check if the host is compromised and update the access level of the host to ROOT	0.000021	Check if the host is compromised and **execute privilege escalation via a `pkexec` vulnerability**	13.174906

Service Scan. In the context of the 'tiny' scenario, the target service that needs to be determined is SSH. To use the Service Scan action, the target host value must be provided, while the port value is optional. For SSH, we provide port 22 in order to minimize the time spent scanning unused ports. When the Service Scan action is executed, it scans the specified host for open ports and matches them to the assigned port value, if it exists. The observations after executing the service scan (Algorithm 1) are equivalent in the NASim and PenGym environments, as follows:

- **NASim**: `{'ssh': 1.0}`, where 1.0 means `True`
- **PenGym**: `['ssh']`

OS Scan. For the OS scan in the 'tiny' scenario, the target operating system that needs to be determined is Linux. To minimize execution time, only port 22 is provided as an optional value for scanning. If the corresponding ports are open and match the assigned port value, a list of the ascertained operating systems is returned. The observations after executing the OS scan (Algorithm 2) are equivalent in the NASim and PenGym environments (details were not included due to space limitations).

Subnet Scan. PenGym uses Nmap to execute subnet scans to identify active hosts within a subnet, and returns a list with the IP addresses of all the interfaces of those active hosts. Using the subnet scan execution on Host(1,0) as an example, the observations after running the action (Algorithm 3) for the 'tiny' scenario are equivalent in the NASim and PenGyn environments, as follows:

- **NASim**: `discovered={(1, 0): True, (2, 0): True, (3, 0): True}`
- **PenGym**:
  `['192.168.122.1', '192.168.122.2', '192.168.123.1', '192.168.123.2', '192.168.123.3', '192.168.124.1', '192.168.124.4', '192.168.125.1', '192.168.125.3', '192.168.125.4']`

The subnet scan function in Nmap can be performed with different arguments, which greatly affect the execution time. Using only the basic `-sS` argument takes about 100 seconds to complete. However, by using the `-Pn` option to skip host discovery, the `-n` option to disable DNS resolution, and the `-T5` timing template, the scan time can be reduced to about 10 seconds. In addition, the `-minparallel` and `-maxparallel` options can be used to control the number of concurrent connections. When all the optimization arguments in Algorithm 3 are used, the scan takes only 2.49 seconds. This shows the importance of carefully selecting the command arguments when performing such time-consuming actions.

Exploit. The PenGym exploit action employs an SSH dictionary attack with a username and password list to gain access to the target host, which is implemented via Metasploit. If the exploit is successful, a Metasploit shell object is returned together with the determined access level: USER or ROOT. This shell object can be used for subsequent actions, such as process scanning and privilege escalation, to investigate and exploit potential vulnerabilities. The observations after executing the Exploit action (Algorithm 4) on the hosts in the 'tiny' scenario are equivalent in NASim and PenGym:

- **NASim**: `access=USER`
- **PenGym**: `shell` object, `access=USER`

Process Scan. Process scanning is implemented by executing the `ps` command that scans for active processes on the target host. This command is executed via the shell previously obtained on the targeted host through an exploit, and returns a list of running processes that are consistent with the scenario description. The observations after executing the Process Scan action are equivalent in NASim and PenGym (details were not included due to space limitations).

Privilege Escalation. Privilege escalation is implemented by attacking vulnerabilities in the `pkexec` package via Metasploit to gain root access on a target host. This requires having a `pkexec` package with a version lower than 0.12 and the Ubuntu 20.04 TLS operating system. The observations after executing the Privilege Escalation action (Algorithm 5) are equivalent in NASim and PenGym (details were not included due to space limitations).

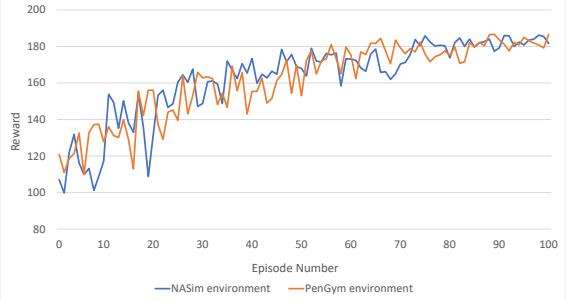

Figure 3: The average reward versus episode number for an RL agent trained in the NASim and PenGym environments.

5.3 RL Agent Experiments

We conducted a series of experiments to demonstrate the feasibility and effectiveness of PenGym. For this purpose, we selected NASim for comparison to showcase the key differences between the emulation approach of PenGym and the simulation approach of NASim. The RL agents were trained and tested using both NASim and PenGym as discussed below.

Agent Training. First, we independently trained 5 agents for each of the NASim and PenGym environments. Each training consisted of 100 episodes, resulting in a total of 5 trained agents for each environment. We used the tabular, epsilon-greedy Q-learning algorithm with experience replay (Sutton and Barto, 2018) in NASim, since preliminary experiments have shown that it has superior performance compared to the other sample algorithms included.

A slightly modified 'tiny' scenario was used to create the training environment in this experiment. Thus, the probability attribute of the `e_ssh` action (Exploit), was set to 0.999999; this provides a scenario that is closer to our cyber range, where the exploit does not fail (note that we used this value because the value 1.0 could not be assigned due to a NASim implementation quirk).

Figure 3 shows the average rewards obtained by training the 5 RL agents in each of the NASim and PenGym environments over 100 episodes. According to our results, the PenGym framework has shown comparable results in training agents in terms of rewards. As for and training time, PenGym took on average 113.12 s compared to 45.12 s for NASim. Although the training time in PenGym is longer, it is still within reasonable limits, especially considering that training is done in a cyber range, not via simulation.

Agent Testing. Next, the performance of the 5 trained agents for each of the NASim and PenGym

environments was evaluated by conducting 20 experiments with the 'tiny' scenario in which the trained agents were used to conduct pentesting in either the NASim or PenGym environments. In addition, the brute force and random agents made available in NASim were also included in the evaluation, undergoing 20 trials in both environments as well.

The experiment results are summarized in Table 3. The results of the 20 trials were averaged to derive the final performance metrics for the agents in each environment. To assess the consistency and variability of the results, we also computed the standard deviation for each test case to provide insights into the stability and reliability of each agent performance across different trials. The results show that all the 5 trained agents in both the NASim and PenGym environments were able to successfully reach the pentesting goal in each of the 20 test trials, resulting in a 100% success rate for all the agents.

The results show that agents trained in NASim required more steps to reach the goal (12.42 and 11.95 steps) than the agents trained in PenGym (7.99 and 7.72 steps). This can be explained by the fact that agents trained in NASim rely on probabilistic values to model action success/failure. Thus, such actions can fail due to randomly generated values that vary each time the action is executed. In contrast, agents trained in PenGym execute real actions in the environment, leading to success based on realistic conditions. Moreover, the agents trained in PenGym demonstrated very good performance, with an average of 7.79 steps compared to the optimal 6 steps of the 'tiny' scenario. The relatively poor performance of the agents trained in NASim arises from the presence of a poorly trained agent, which reduced the average performance of those agents during testing. Additionally, testing in PenGym resulted in slightly better performance than testing in NASim, with 7.99 steps compared to 7.72 steps, for example. As for the brute force and random agents, they also exhibited comparable performance in the NASim and PenGym environments, but their results were far from optimal.

The main advantage of PenGym lies in its ability to facilitate training with real actions in an actual environment. This eliminates the need for modeling actions using execution assumptions and success probabilities, as observations come from the execution of real actions. By executing actions without considering probabilistic factors, agents trained in PenGym improve their algorithm based on realistic results. For example, the success of an SSH-based exploit action executed in NASim is determined by a random value. This means it may fail at unexpected times, which does not accurately reflect realistic actions, and leads to non-deterministic effects on agent learning—which does not happen when using PenGym. As the system grows, the action space becomes larger, and these probabilistic factors may affect the learning process of the agents in unknown ways. Therefore, our approach ensures a more realistic representation of the security dynamics in the network.

However, one drawback of PenGym is its longer execution time compared to the simulation-based NASim environment, since actions are physically executed in the network. Nonetheless, we consider the extended training duration reasonable given that the training takes place in a realistic cyber range. Moreover, the full automation ensures that the execution time should be lower than human-based pentesting and red team exercises, which often take weeks or even months (Li et al., 2022).

5.4 Additional RL Agent Experiments

To demonstrate the potential of PenGym in adapting to more complex network environments, we also conducted several additional experiments using higher complexity scenarios, such as 'tiny-hard', 'tiny-small', 'small-honeypot' and 'small-linear'.

For example, the 'small-linear' scenario consists of 6 subnets and 8 hosts, and its action space includes a total of 72 available actions. Most hosts have several services and processes, which correspond to multiple service-based and process-based actions within the host. To increase the challenge of the scenario, some hosts do not have any processes, and the number of firewalls is also higher.

We trained an agent using the same algorithm mentioned in Section 5.3 for both the NASim and PenGym environments. The training process involved 500 epochs for the 'small-linear' scenario, and 150 epochs for the other scenarios, which was enough to allow the agent performance to stabilize.

According to the results in Table 4, PenGym training took approximately two to three times longer than for NASim. In the simple 'tiny' scenario, the ratio between PenGym and NASim was around 2.5. However, as the network size and complexity increased, the ratio decreased to less than 2.5. Thus, the training time ratio was 1.7 for the 'tiny-small' scenario, 1.4 for the 'small-honeypot' scenario, and 2.2 for the 'small-linear' scenario. This suggests that more complex scenarios lead to a lower comparative training time in PenGym, a property that we attribute to the more significant effect of our optimization methods when a larger number of actions is repeated.

Table 4 also shows the total number steps that were taken to reach the pentesting goal. These results

Table 3: Comparison of the execution performance of the RL agents trained and tested in the NASim and PenGym environments; brute force and random agents are also included for reference.

Agent	Training	Testing	Success Rate	Avg. Steps	Avg. Exec. Time [s]	Std. Dev.
RL	NASim	NASim	20/20	12.42	0.0025	6.081
RL	NASim	PenGym	20/20	11.95	45.8329	5.046
RL	PenGym	NASim	20/20	7.99	0.0016	0.229
RL	PenGym	PenGym	20/20	7.72	43.1357	0.219
Brute force	N/A	NASim	20/20	47.00	0.0022	0.000
Brute force	N/A	PenGym	20/20	47.00	58.0670	0.000
Random	N/A	NASim	20/20	102.30	0.0048	38.134
Random	N/A	PenGym	20/20	94.80	62.0402	53.121

Table 4: Comparison of the training time and step count of the RL agents trained in the NASim and PenGym environments for several additional scenarios.

Scenario	Training	Time [s]	Steps
tiny	NASim	45.12	15
	PenGym	113.12	11
tiny-hard	NASim	68.49	8
	PenGym	198.75	13
tiny-small	NASim	316.99	9
	PenGym	539.35	10
small-honeypot	NASim	1680.45	14
	PenGym	2456.82	8
small-linear	NASim	2091.45	17
	PenGym	4771.25	24

are recorded from the last training epoch for reference purposes, so they do not necessarily indicate the overall performance of those agents. Overall, these experiments demonstrate the positive characteristics of PenGym in terms of training performance when dealing with more complex scenarios.

6 DISCUSSION

This paper thoroughly demonstrated the use of a cyber range based on the 'tiny' scenario in NASim for realistic RL agent training. However, this scenario has certain limitations, such as small size and simple scope, that prevent it from being an accurate reflection of real-world circumstances. As a result, agent performance in actual cyberattacks may be overestimated when using this scenario, masking issues that would become apparent in more complex settings. Nevertheless, the 'tiny' scenario includes all the basic necessary components of a real environment, such as hosts, subnets, firewalls, and connections between subnets. Therefore, the use of this scenario to demonstrate the feasibility and potential of utilizing the real PenGym environment in comparison with simulation environments is fully justified. To further increase the effectiveness and generality of PenGym, more complex network environments need to be recreated.

In addition, the evaluation of the trained agents when conducting pentesting demonstrates the effectiveness of PenGym, with the PenGym trained agents achieving a slightly lower average step count compared to NASim trained ones. Moreover, the average pentesting execution times were acceptable, being less than about 45 s for the trained RL agents.

Furthermore, in the 'tiny' scenario privilege escalation is supposed to be achieved via the Tomcat process. However, to the best of our knowledge, it is not possible to gain root access directly by attacking vulnerabilities in the Tomcat process, at least in recent OSes. Instead, the Tomcat process can be used as an intermediate step to obtain a Meterpreter shell, then root access can be gained by exploiting local operating system vulnerabilities through Meterpreter. Such an implementation would eliminate the need to first exploit the host to get access to it, so it would change the built-in sequence of actions in NASim. To address this issue, in PenGym we used instead a new method for privilege escalation, which uses a vulnerability in the `pkexec` package to gain administrative access to the target hosts, as previously mentioned. This modification enables PenGym to follow a realistic attack scenario that is still in accordance with NASim assumptions, which was important to do currently to make their comparison possible.

We envision that PenGym could be integrated into current cybersecurity practices, since RL agents trained via PenGym can mimic attacker strategies and actions in a realistic manner. This integration would make possible the automation of pentesting in the future, as RL agents trained via PenGym could assist pentesters in their work, further improving the effectiveness of cybersecurity assessment.

7 CONCLUSION

In this paper, we introduced PenGym, a framework for creating real environments to support training RL agents for penetration testing purposes. By enabling the execution of real actions with real observations, PenGym eliminates the need for the probabilistic modeling of actions, resulting in a more accurate representation of security dynamics compared to simulation-based environments. Since PenGym agents are trained in a cyber range, they experience actual network conditions. This approach potentially enhances the applicability of the trained agents when they are deployed in a real-world infrastructure.

The framework has been validated and refined through several experiments, demonstrating the correctness of the action implementation, as well as the fact that PenGym can provide reliable results and reasonable execution times for training. Although the PenGym training times are longer than those of simulation environments (e.g., 113.12 s versus 45.12 s on average in our experiments when using the 'tiny' scenario), we consider that this value is reasonable given the actual cyber range and network use.

The PenGym framework was released on GitHub (`https://github.com/cyb3rlab/PenGym`) as open source. We plan to add support for more complex environments, thus enabling users to simulate more realistic and larger scenarios that would make possible more comprehensive testing and analysis.

In addition, we aim to improve the cyber range creation method, which is currently mainly a manual process that is time-consuming and cumbersome. Our next goal is to automate cyber range creation by leveraging the functionality of the open-source CyTrONE framework (Beuran et al., 2018), so that the creation process is streamlined and more efficient.

REFERENCES

Beuran, R., Tang, D., Pham, C., Chinen, K., Tan, Y., and Shinoda, Y. (2018). Integrated framework for hands-on cybersecurity training: CyTrONE. *Computers & Security*, 78C:43–59.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. *arXiv:1606.01540*.

Chaudhary, S., O'Brien, A., and Xu, S. (2020). Automated post-breach penetration testing through reinforcement learning. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–2. IEEE.

Furfaro, A., Piccolo, A., Parise, A., Argento, L., and Sacca, D. (2018). A cloud-based platform for the emulation of complex cybersecurity scenarios. *Future Generation Computer Systems*, 89:791–803.

Ghanem, M. C. and Chen, T. M. (2018). Reinforcement learning for intelligent penetration testing. In *2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pages 185–192.

Li, L., El Rami, J.-P. S., Taylor, A., Rao, J. H., and Kunz, T. (2022). Enabling a network AI gym for autonomous cyber agents. In *2022 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 172–177. IEEE.

Lyon, G. (2014). Nmap security scanner. https://nmap.org/.

Maynor, D. (2011). *Metasploit toolkit for penetration testing, exploit development, and vulnerability research*. Syngess Publishing, Elsevier.

McInerney, D. (2020). Pymetasploit3. https://pypi.org/project/pymetasploit3/.

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.

Molina-Markham, A., Miniter, C., Powell, B., and Ridley, A. (2021). Network environment design for autonomous cyberdefense. *arXiv:2103.07583*.

National Vulnerability Database (2021). CVE-2021-4034. https://nvd.nist.gov/vuln/detail/CVE-2021-4034. Accessed: May 2, 2023.

Norman, A. (2021). Python-nmap. https://pypi.org/project/python-nmap/.

Pozdniakov, K., Alonso, E., Stankovic, V., Tam, K., and Jones, K. (2020). Smart security audit: Reinforcement learning with a deep neural network approximator. In *2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, pages 1–8.

Schwartz, J. and Kurniawati, H. (2019). Autonomous penetration testing using reinforcement learning. *arXiv:1905.05965*.

Standen, M., Lucas, M., Bowman, D., Richer, T. J., Kim, J., and Marriott, D. (2021). Cyborg: A gym for the development of autonomous cyber agents. In *Proceedings of the 1st International Workshop on Adaptive Cyber Defense*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

The MITRE Corporation (2018). Brawl. https://github.com/mitre/brawl-public-game-001.

Zhou, S., Liu, J., Hou, D., Zhong, X., and Zhang, Y. (2021). Autonomous penetration testing based on improved deep Q-network. *Applied Sciences*, 11(19).