IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 14, No. 5, October 2025, pp. 4061~4073
ISSN: 2252-8938, DOI: 10.11591/ijai.v14.i5.pp4061-4073

Design and analysis of reinforcement learning models for automated penetration testing

Suresh Jaganathan[1], Mrithula Kesavan Latha[2], Krithika Dharanikota[3]
[1]Department of Computer Science and Engineering, Sri Sivasubramaniya Nadar College of Engineering, Chennai, India
[2]Software Development Analyst, Citicorp Services India Ltd, Chennai, India
[3]Department of Computer Science, University of Southern California, Los Angeles, United States

Article Info

Article history:

Received Jun 13, 2024
Revised Jul 26, 2025
Accepted Aug 6, 2025

Keywords:

Asynchronous episodic deep deterministic policy gradient
Automated penetration testing
Deep deterministic policy gradient
Deep Q-network
Reinforcement learning

ABSTRACT

Our paper proposes a framework to automate penetration testing by utilizing reinforcement learning (RL) capabilities. The framework aims to identify and prioritize vulnerable paths within a network by dynamically learning and adapting strategies for vulnerability assessment by acquiring the network data obtained from a comprehensive network scanner. The study evaluates three RL algorithms: deep Q-network (DQN), deep deterministic policy gradient (DDPG), and asynchronous episodic deep deterministic policy gradient (AE-DDPG) in order to compare their effectiveness for this task. DQN uses a learned model of the environment to make decisions and is hence called model-based RL, while DDPG and AE-DDPG learn directly from interactions with the network environment and are called model-free RL. By dynamically adapting its strategies, the framework can identify and focus on the most critical vulnerabilities within the network infrastructure. Our work is to check how well the RL technique picked security vulnerabilities. The identified vulnerable paths are tested using Metasploit, which also confirmed the accuracy of the RL approach's results. The tabulated findings show that RL promises to automate penetration testing tasks.

Corresponding Author:

Suresh Jaganathan
Department of Computer Science and Engineering, Sri Sivasubramaniya Nadar College of Engineering
Chennai, India
Email: sureshj@ssn.edu.in

1. INTRODUCTION

Penetration testing, or pen-testing, is crucial for evaluating information technology (IT) infrastructure resilience against cyber threats by proactively identifying vulnerabilities. This process helps organizations strengthen their security posture and prevent potential data breaches and financial detriment. Pen-testing also aids in regulatory compliance and provides insights for informed security investments which mandate regular security evaluations. With the evolution of machine learning, particularly reinforcement learning (RL), pen-testing techniques are becoming more automated and effective, improving security assessments for organizations. We will examine the existing landscape of model-based penetration testing and introduce an innovative framework [1] that harnesses RL to streamline and augment the penetration testing process.

Combining penetration testing with RL presents a promising strategy to enhance conventional methodologies. RL algorithms can automatically prioritize vulnerabilities based on their likelihood and potential impact, minimizing the need for manual intervention. These algorithms adapt effectively to dynamic environments and evolving threat landscapes, ensuring that penetration testing techniques remain effective over time. Furthermore, RL enables parallel exploration of various attack strategies, leading to more comprehensive vulnerability assessments. This scalability makes RL well-suited for handling extensive and intricate networks, providing organizations with a compre-hensive approach to security evaluations.

Penetration testing primarily relies on model-based techniques, where experts create a detailed model of the network using data from scans and analyses to identify vulnerabilities. While effective in controlled environments, this approach struggles with the dynamic nature of modern networks. The emergence of new vulnerabilities and network changes make maintaining an accurate model challenging. To address this, integrating RL into model-based methodologies can provide adaptive capabilities. By doing so, organizations can improve the accuracy and effectiveness of their vulnerability assessments, leading to robust security measures.

Penetration testing methods have evolved, yet obstacles persist in efficiently detecting and prioritizing vulnerabilities, especially across extensive, intricate networks. Manual processes demand substantial time and effort, potentially resulting in oversight or incorrect prioritization of security vulnerabilities due to human fallibility. Additionally, the complexity of modern networks can make it difficult to identify the most critical vulnerabilities that pose the greatest risk to an organization. Traditional model-based approaches struggle to adapt to new problems, resulting in less effective vulnerability assessments.

The proposed solution addresses challenges in penetration testing by developing an automated framework using RL to identify and prioritize vulnerabilities in a network. The framework dynamically learns and adapts its strategies based on network data from a comprehensive scanner. It consists of three modules: network analyzer, RL engine using deep Q-network (DQN), deep deterministic policy gradient (DDPG), and asynchronous episodic deep deterministic policy gradient (AE-DDPG) algorithms, and a pen-testing module. The network analyzer collects and analyses network data, identifying vulnerabilities and attack paths. The RL engine prioritizes vulnerabilities and determines optimal attack paths, while the pen-testing module verifies these paths using industry-standard tools. This integration aims to automate and optimize penetration testing, adapting to network changes and reducing cyberattack risks. Ultimately, the framework seeks to revolutionize cybersecurity practices, providing defenders with adaptive and intelligent tools to combat cyber threats effectively.

2. RELATED WORKS

The concept of automating penetration testing [2] has been a longstanding pursuit, initially manifesting in the form of attack graphs [3]. Attack graphs [4] serve as models to depict systems and their susceptibility to particular exploits. Traditionally, identifying these attack paths involved employing classical planning methodologies. However, a substantial disadvantage of this approach is its reliance on comprehensive knowledge of the network topology and the configuration of each machine, rendering it impractical from the perspective of an attacker.

An alternative approach to modelling and strategizing attacks against a system involves employing a Markov decision process (MDP) to simulate the operational environment. An MDP [5] serves as a versatile framework for representing discrete decision-making scenarios amid uncertainty. When applied to penetration testing, the state space within the MDP encompasses the potential configurations of target machines or the network, with actions representing available exploits or scans and rewards contingent upon the costs associated with actions and the value accrued upon successfully compromising a system. RL emerges as a technique capable of deriving optimal policies for MDPs. It leverages interactions with the environment to generate samples, thereby optimizing performance. Distinct advantages over classical planning methodologies include its adeptness at handling expansive environments and its applicability in scenarios where either a model of the environment is absent or utilizing the model proves computationally intractable.

A partially observable Markov decision process (POMDP) is a variant of the MDP that incorporates uncertainty about the precise state of the system. In a POMDP, the current state is modelled as a probability distribution encompassing all potential states. When applied to penetration testing, the state space within the POMDP encompasses the potential configurations of the target machine or network. Actions represent available exploits or scans, while the observation space comprises the information gathered when an exploit or scan is executed (e.g., open ports, success, or failure of the exploit). Rewards are determined based on the cost of an action and the value gained from successfully compromising a system. POMDP leverages the concept of "belief" to represent uncertainty in decision-making during attacks. POMDP is used to simulate [6] the "Pentest" task, aiming to find the shortest path to the target node. However, due to the inherent complexity of POMDP algorithms, their applicability is confined to environments involving only two hosts. Recognizing the limitations posed by POMDP algorithms, intelligent automated penetration testing system (IAPTS) [7] is developed to address the challenge of automating pen-testing in large network environments.

Despite its utility, POMDP-based solvers encounter a critical limitation in scaling efficiently. As the state space increases in size, POMDP-based solvers often become computationally infeasible, hampering their practical application in large-scale scenarios. Consequently, an increasing number of researchers are opting to model the pen-test task using the MDP, where action outcomes are deterministic, thereby circumventing the computational complexities associated with POMDP-based approaches.

While model-based methods have proven effective, they are inherently constrained by the requirement for human experts to define the dynamics of the models. In recent times, there has been a shift towards employing model-free RL algorithms to address penetration testing challenges. Unlike model-based approaches reliant on expert-designed models, model-free agents autonomously interact with the environment to derive optimal strategies.

Our research adopts a similar model-free RL approach, albeit with a primary focus on critically assessing both model-based and model-free RL techniques and evaluating their efficacy in automating penetration testing processes. A framework [8] is proposed for using RL to learn attack paths, which is evaluated on a simulated environment and showed that it can learn to find effective attack paths more efficiently than traditional penetration testing methods. Then comes the traditional DQN algorithm. Although it incorporates RL, it easily overestimates the Q value, which leads to ineffective policy updates and unstable behaviour [9]. The DDPG algorithm, as presented in [10], offers a model-free, off-policy actor-critic approach employing deep function approximators capable of learning policies in high-dimensional, continuous action spaces.

By integrating insights from the success of DDPG, AE-DDPG [11] addresses the challenge of sample imbalance. Additionally, the AE-DDPG algorithm introduces episodic memory into deep RL techniques for continuous problems, leveraging episodic control (EM) thinking to redesign the experience replay of DDPG, thereby facilitating rapid acquisition of high-reward policies. Notably, AE-DDPG is the first model to incorporate episodic memory into deep-reinforcement learning (DRL) techniques for continuous problems. It also incorporates multiple agents [12] which interact with the environment asynchronously in [13], a comprehensive examination of DRL challenges and corresponding solutions is presented, particularly focusing on the reward design issue within human-robot collaboration contexts. Furthermore, the study explores potential avenues for future research within this domain.

The multi-dimensional deep Q-network (MDDQN) algorithm, introduced by Chen [14], integrates attack graphs from multihost, multistage vulnerability analysis language (MulVAL) with double deep Q-network (DDQN) to enhance attack path planning. Following this, MulVAL [15] and depth-first search (DFS) were utilized to construct an attack matrix, with DQN employed to analyze the matrix and identify the most vulnerable attack path for the network. This identified attack path could then undergo testing using industry-standard tools such as Metasploit and Wireshark, as outlined in [16]. To elucidate the usage of the Metasploit framework tool in a detailed manner, outlining procedures for each testing phase along with the requisite commands (syntax) conducted within a Kali Linux environment [17]. Additionally, an innovative approach to automated penetration testing employing DRL is presented in a study by [18]. This framework utilizes DRL techniques to identify optimal attack paths within simulated network environments, leveraging tools such as network mapper (Nmap), MulVAL, and the national vulnerability database (NVD) to analyze attack graphs and determine the most effective path based on common vulnerability scoring system (CVSS) scores [19]. This paper significantly enhances our understanding of network analysis tools and methodologies. Given the capabilities of these algorithms, our research focuses on comparing DQN as a representative of model-based learning with DDPG and AE-DDPG for model-free learning in the context of automating penetration testing [20].

3. METHOD

Figure 1 shows the proposed architectural framework which is divided into three distinct modules: i) network scanning and information gathering, ii) RL, and iii) pen-testing. A detailed examination of each module, including their respective inputs and outputs, is provided below in order to impart a comprehensive understanding of their roles within the overall architecture. The framework collects user input on the logical target network, including vulnerability information. Next, prospective attack trees are identified using the MulVAL attack-graph generator and fed into the RL engine in a reduced format. The user can then examine how the attack could be executed on an actual target network by using penetration testing tools like Metasploit to leverage the attack paths produced by the RL engine in the framework.

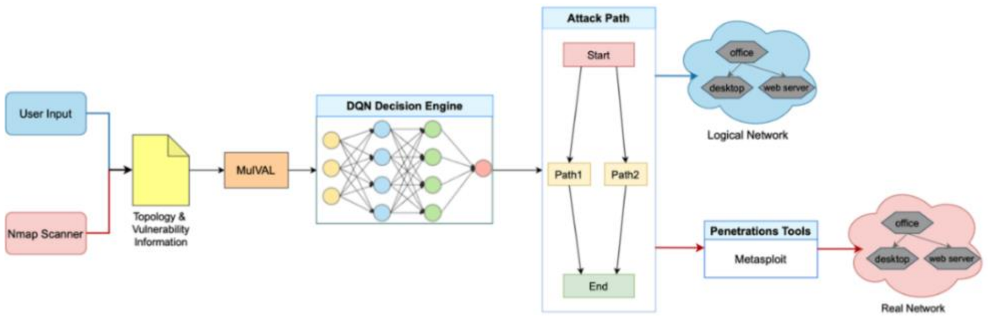


Figure 1. Architecture diagram of the proposed framework

3.1. Network scanning and information gathering

Our research employs MulVAL [21] as the network analyzer. MulVAL is a logic-based network security analyzer that is accessible as an open-source tool designed to construct an attack graph for a specified network architecture. All experiments in our study are conducted within a simulated network environment. The input, comprising network information, is expressed in Datalog. MulVAL processes this network data, producing outputs that include the identification of vulnerabilities and machine configuration details presented in the form of predicates. The vulnerabilities are compared with the common vulnerabilities and exploits (CVE) in the NVD, which is the U.S. government's repository of vulnerability management data based on National Institute of Standards and Technology (NIST) standards. We are also using the Metasploit Exploit Database, which contains a list of vulnerabilities that have been discovered. These vulnerabilities are identified and analyzed by MulVAL through the integration of formal vulnerability specifications from bug-reporting communities, host and network configuration information, and other relevant data encoded as Datalog facts. MulVAL's reasoning engine is meant to scale efficiently with network size, enabling efficient analysis of networks containing thousands of machines.

3.2. Reinforcement learning engine

Following the analysis conducted by MulVAL, the outputs, encompassing identified vulnerabilities and machine configuration details in the form of predicates, serve as crucial inputs for our RL model. The RL model is tasked with determining the most vulnerable paths within the network topology, employing advanced algorithms such as DQN, DDPG and AE-DDPG. These algorithms undergo a comprehensive evaluation to discern their effectiveness in prioritizing and navigating potential vulnerabilities within the network and enhance the efficiency of automated penetration testing [22]. Figure 2 depicts the outline of the RL engine.

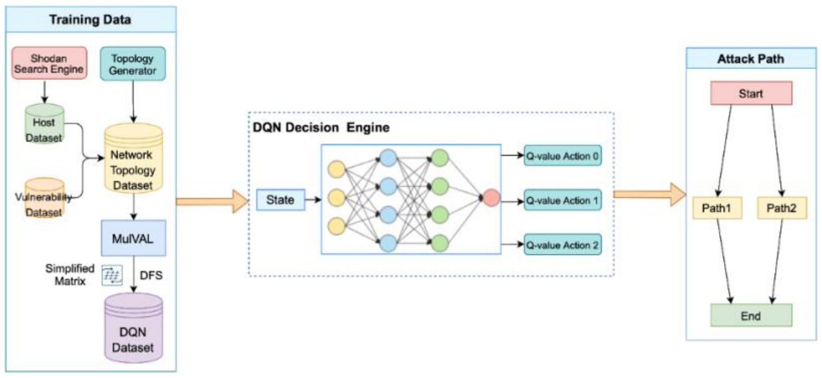


Figure 2. Outline of the RL engine

3.2.1. Deep Q-network

DQN is a sophisticated deep RL model which achieved extraordinary performance in learning control policies from high-dimensional sensory input. The training DQN implements a variant of the Q-learning algorithm [23], which involves an iterative update of network weights using stochastic gradient descent. DQN, an RL model, poses some difficulties during training due to sparse and delayed rewards, correlated data, and non-stationary distributions. To overcome this, DQN adopts a technique called experience replay, where the agent's experiences, consisting of tuples of (state, action, reward, and next state), are stored in a replay memory. Random uniform sampling of experiences from this memory enables a more efficient use of past experiences, thereby smoothing out the training distribution and improving data efficiency. Additionally, it stabilizes the learning process by reducing the correlation between consecutive samples. Additionally, DQN employs a frame-skipping technique, which reduces the computational demands by repeating the agent's chosen action for a certain number of frames. This technique enables the agent to process more game frames without significantly increasing the computational cost. Furthermore, during training, DQN employs an epsilon-greedy policy to strike a balance between exploration and exploitation.

3.2.2. Deep deterministic policy gradient

The DDPG algorithm is a model-free algorithm that utilizes the deterministic policy gradient to operate in continuous action spaces. The architecture is based on an actor-critic framework with a replay buffer and utilizes a target network to stabilize the learning process. DDPG employs four neural networks: a Q-network, a deterministic policy network, a target Q-network, and a target policy network. The interaction between the Q-network and policy network is very similar to a simple Advantage Actor-Critic. However, in DDPG, the actor directly maps states to actions rather than producing a probability distribution across a discrete action space. The target networks are time-delayed duplicates of their original networks that gradually follow the taught networks. The use of target value networks significantly increases learning stability.

DDPG utilizes a replay buffer to sample experiences and update the neural network parameters. Throughout each trajectory roll-out, all experience tuples (state, action, reward, and next_state) are saved and maintained in a finite cache called a "replay buffer." Random mini-batches of experience from the replay buffer are then sampled while the value and policy networks are updated. The value network is updated in a manner similar to Q-learning. The target value network and target policy network, on the other hand, generate the next-state Q-values, which are then utilized to minimize the mean-squared loss between the updated Q-value and the original Q-value. The policy function seeks to maximize the expected return and compute the policy loss, so the derivative of the objective function with respect to the policy parameter is used. Exploration in continuous action spaces involves adding noise to both the action and the parameter space. The noise is added to the actor policy to allow for exploration independent of the learning procedure, and it is created using the Ornstein-Uhlenbeck process to offer temporally correlated exploration.

3.2.3. Asynchronous episodic deep deterministic policy gradient

The AE-DDPG algorithm is developed for continuous control in computationally complex environments. The algorithm comprises an actor-critic duo, where the actor interacts with multiple stochastic environments simultaneously to collect data asynchronously. The collected data is stored in memory buffers for experience replay, with a focus on balancing data generation and utilization to improve sample efficiency and diversity. AE-DDPG distinguishes itself from DDPG by addressing data insufficiency and training inefficiency through an asynchronous framework, episodic control, and the injection of new noise in the action space, which ultimately leads to improved learning efficiency and performance in complex environments. The architecture of AE-DDPG includes memory buffers for experience replay, with separate cache buffers for individual interaction threads and two memory buffers for experience replay. This algorithm incorporates the concept of episodic control to swiftly acquire advanced knowledge from high-reward experiences while also increasing the diversity of sampling paths for experience replay.

In terms of algorithm functionality, asynchronous interaction enables the actor to collect more data for policy learning, particularly in computationally complex environments. The memory buffers store trajectories for experience replay, utilizing a novel bio-inspired episodic experience replay approach that aims to balance data generation and utilization speeds to prevent sample imbalance and enhance sample diversity. In addition, a novel type of noise called random walk noise has been developed to enhance exploration efficiency and sample diversity in RL.

3.3. Pentesting

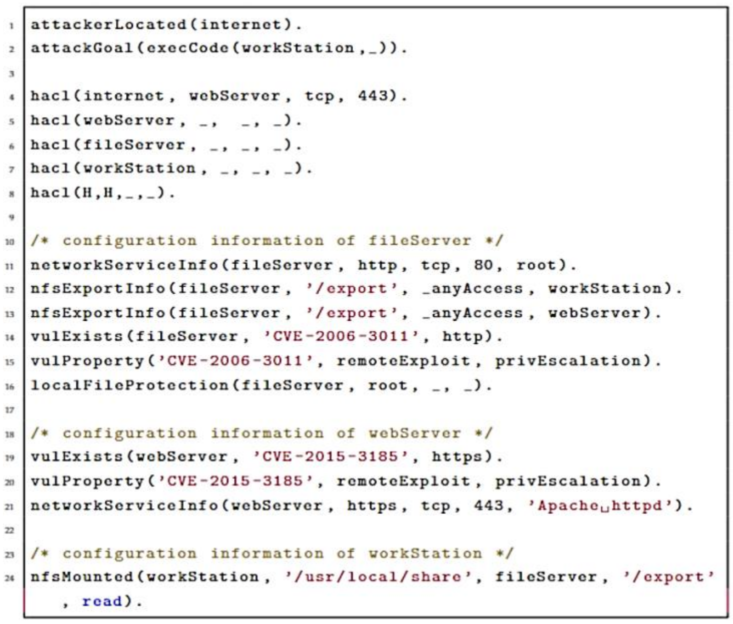
Subsequently, to validate the identified vulnerable paths prioritized by RL, we utilize established penetration testing tools, like Metasploit. Metasploit aids in assessing the real-world exploitability of vulnerabilities [24], [25]. This comprehensive validation approach ensures the practicality and reliability of the RL-driven results, offering a thorough assessment of potential security weaknesses within the network. By integrating these robust penetration testing tools into our validation process, we aim to rigorously evaluate the effectiveness and reliability of the RL-driven results, ensuring that the identified vulnerabilities align with real-world scenarios and enhancing the overall security posture of the network.

4. RESULTS AND DISCUSSION

The initial phase encompasses the development of the network scanning and information-gathering module, which uses MulVAL to generate attack graphs predicated on simulated network conditions. These graphs subsequently serve as inputs for our RL engine.

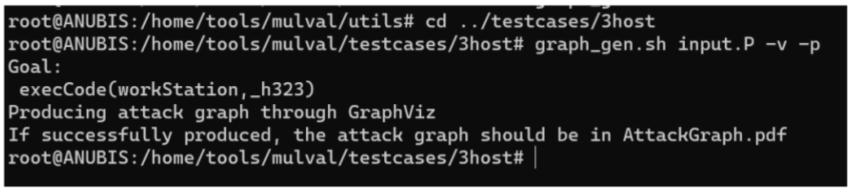
4.1. Multistage vulnerability analysis language

The initial phase encompasses the development of the network scanning and information-gathering module. The central aim of this phase was the seamless integration of MulVAL into our project framework. This integration facilitated the generation of attack graphs predicated on simulated network conditions, which subsequently serve as inputs for our RL engine. Prerequisites for the installation of MulVAL include XSB, GraphViz, and MariaDB, an open-source-compatible version of MySQL. These components constitute essential dependencies for MulVAL's proper functioning. Figure 3 presents the input Datalog code furnished to MulVAL for network scanning and information gathering and Figure 4 shows implementation of MulVal.



```
attackerLocated(internet).
attackGoal(execCode(workStation,_)).

hacl(internet, webServer, tcp, 443).
hacl(webServer, _,  _, _).
hacl(fileServer, _, _, _).
hacl(workStation, _, _, _).
hacl(H,H,_,_).

/* configuration information of fileServer */
networkServiceInfo(fileServer, http, tcp, 80, root).
nfsExportInfo(fileServer, '/export', _anyAccess, workStation).
nfsExportInfo(fileServer, '/export', _anyAccess, webServer).
vulExists(fileServer, 'CVE-2006-3011', http).
vulProperty('CVE-2006-3011', remoteExploit, privEscalation).
localFileProtection(fileServer, root, _, _).

/* configuration information of webServer */
vulExists(webServer, 'CVE-2015-3185', https).
vulProperty('CVE-2015-3185', remoteExploit, privEscalation).
networkServiceInfo(webServer, https, tcp, 443, 'Apache␣httpd').

/* configuration information of workStation */
nfsMounted(workStation, '/usr/local/share', fileServer, '/export'
   , read).
```

Figure 3. Datalog code for a 3-host network to be fed to MulVAL



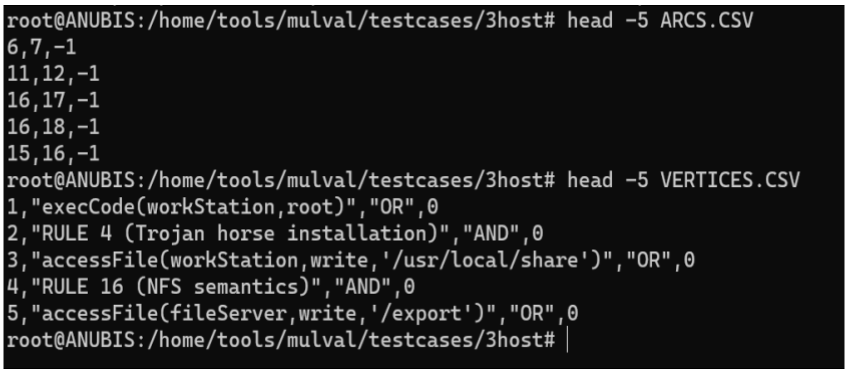
```
root@ANUBIS:/home/tools/mulval/utils# cd ../testcases/3host
root@ANUBIS:/home/tools/mulval/testcases/3host# graph_gen.sh input.P -v -p
Goal:
 execCode(workStation,_h323)
Producing attack graph through GraphViz
If successfully produced, the attack graph should be in AttackGraph.pdf
root@ANUBIS:/home/tools/mulval/testcases/3host# |
```

Figure 4. Implementation of MulVal

Firstly, we establish the location of the attacker, identifying them as situated within the "internet" entity. Additionally, we specify the attack goal of the attacker, indicating their intent to execute code on the "workStation" entity. Next, we define rules regarding network access, denoted by the "hack" predicate. These rules permit access between various entities within the network, such as the "webServer," "fileServer," and "workstation." The configuration information for network entities, including the "fileServer," "webServer," and "workStation," is then detailed. This information encompasses network service details, vulnerability identifiers, and network file system (NFS) configurations. For instance, the configuration information for the "fileServer" entity specifies that it runs the "mountd" service over the remote procedure call (RPC) protocol on port 100005, granting root privileges. Furthermore, NFS export information indicates that the "/export" directory on the file server is accessible with any permissions by the "workStation."

Vulnerability information is also provided, identifying vulnerabilities associated with the "fileServer" and "webServer" entities. Properties of these vulnerabilities, such as their exploitability and potential impact, are outlined within the code. This code is now furnished as input to MulVAL. By default, MulVAL outputs the resulting attack graph in both textual (AttackGraph.txt) and XML (AttackGraph.xml) formats, with the intended semantics being self-evident. Furthermore, the invocation of the -v option facilitates the generation of a visual representation of the attack graph in PDF format (AttackGraph.pdf) through GraphViz.

Upon specification of the appropriate options, MulVAL extends its output capabilities to include attack-graph information in CSV format, comprising VERTICES.CSV and ARCS.CSV files. These CSV files are instrumental for subsequent rendering programs to generate diverse views of the attack graph at a later stage. We have also generated the attack graph for another network where there are two more web servers associated with the "httpd" service, marked as a local exploit with the potential impact of a denial of service (DoS) and "webServer3" has a vulnerability with the ID 'VULN-ID-3' associated with the "ftpd" service, marked as a remote exploit with the potential impact of unauthorized access respectively. Figure 5 shows the output of MulVAL and the generated attack graph.



```
root@ANUBIS:/home/tools/mulval/testcases/3host# head -5 ARCS.CSV
6,7,-1
11,12,-1
16,17,-1
16,18,-1
15,16,-1
root@ANUBIS:/home/tools/mulval/testcases/3host# head -5 VERTICES.CSV
1,"execCode(workStation,root)","OR",0
2,"RULE 4 (Trojan horse installation)","AND",0
3,"accessFile(workStation,write,'/usr/local/share')","OR",0
4,"RULE 16 (NFS semantics)","AND",0
5,"accessFile(fileServer,write,'/export')","OR",0
root@ANUBIS:/home/tools/mulval/testcases/3host# |
```

Figure 5. Execution of MulVAL on 3-host input and terminal output

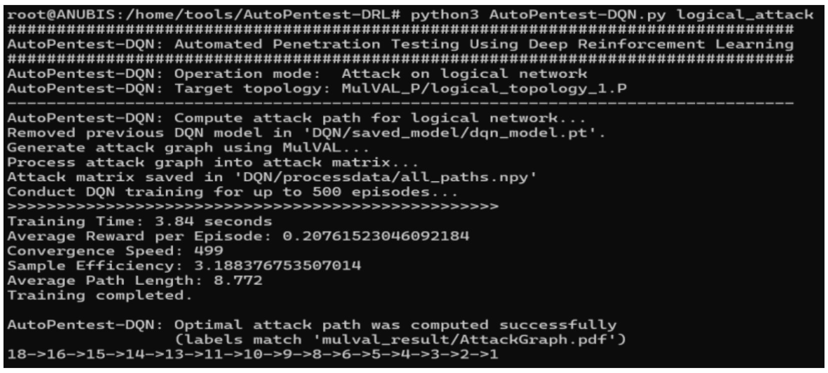
4.2. Reinforcement learning engine

The attack graphs generated by MulVAL serve as crucial inputs for three RL engine models: i) DQN, ii) DDPG, and iii) AE-DEPG. These graphs, encapsulating potential attack vectors and network vulnerabilities, guide the model's exploration process. By analyzing the attack graph, the models can learn the relationships between network entities, vulnerabilities, and potential exploits. Using the learned details, the model can identify and prioritize the most vulnerable path within the network, ultimately leading to the most effective attack strategy for the simulated penetration testing scenario. The most vulnerable path in the attack graph is produced as output refer Figures 6 to 8.

4.3. Performance metrics

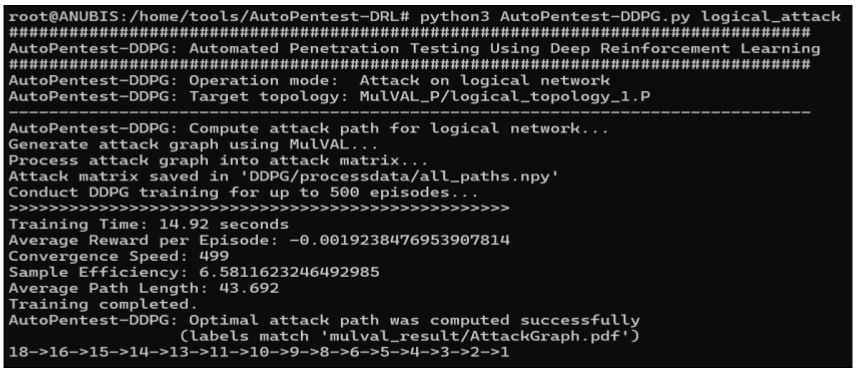
These are the performance metrics used to evaluate and compare the performance of the three algorithms:

– Average reward per episode: this metric measures the average reward obtained by the RL algorithm in each episode of training. A higher average reward indicates that the algorithm is performing better at achieving its objectives.
– Training time (in seconds): this metric measures the time taken for the algorithm to complete training. A shorter training time is desirable as it indicates that the algorithm can learn and adapt more quickly to the environment.
– Convergence speed (in episodes): convergence speed refers to the number of episodes it takes for the algorithm to converge to a stable policy. A lower convergence speed indicates that the algorithm learns more quickly and efficiently.
– Sample efficiency: sample efficiency measures how well the algorithm utilizes the available training data. A higher sample efficiency indicates that the algorithm can achieve good performance with fewer training samples.
– Average path length (APL): in the context of penetration testing, APL refers to the average number of steps or actions taken by the algorithm to reach the target (e.g., identifying and prioritizing vulnerabilities). A shorter APL indicates that the algorithm is able to find more efficient solutions.



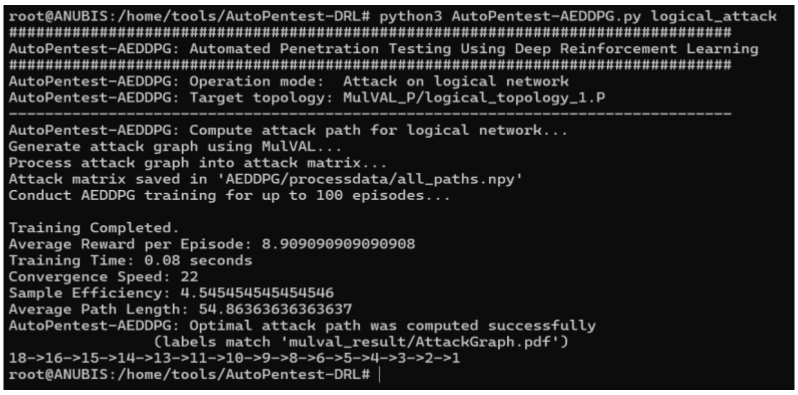
```
root@ANUBIS:/home/tools/AutoPentest-DRL# python3 AutoPentest-DQN.py logical_attack
################################################################################
AutoPentest-DQN: Automated Penetration Testing Using Deep Reinforcement Learning
################################################################################
AutoPentest-DQN: Operation mode:  Attack on logical network
AutoPentest-DQN: Target topology: MulVAL_P/logical_topology_1.P
--------------------------------------------------------------------------------
AutoPentest-DQN: Compute attack path for logical network...
Removed previous DQN model in 'DQN/saved_model/dqn_model.pt'.
Generate attack graph using MulVAL...
Process attack graph into attack matrix...
Attack matrix saved in 'DQN/processdata/all_paths.npy'
Conduct DQN training for up to 500 episodes...
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Training Time: 3.84 seconds
Average Reward per Episode: 0.20761523046092184
Convergence Speed: 499
Sample Efficiency: 3.188376753507014
Average Path Length: 8.772
Training completed.

AutoPentest-DQN: Optimal attack path was computed successfully
                 (labels match 'mulval_result/AttackGraph.pdf')
18->16->15->14->13->11->10->9->8->6->5->4->3->2->1
```

Figure 6. Execution and terminal output of DQN algorithm



```
root@ANUBIS:/home/tools/AutoPentest-DRL# python3 AutoPentest-DDPG.py logical_attack
################################################################################
AutoPentest-DDPG: Automated Penetration Testing Using Deep Reinforcement Learning
################################################################################
AutoPentest-DDPG: Operation mode:  Attack on logical network
AutoPentest-DDPG: Target topology: MulVAL_P/logical_topology_1.P
--------------------------------------------------------------------------------
AutoPentest-DDPG: Compute attack path for logical network...
Generate attack graph using MulVAL...
Process attack graph into attack matrix...
Attack matrix saved in 'DDPG/processdata/all_paths.npy'
Conduct DDPG training for up to 500 episodes...
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Training Time: 14.92 seconds
Average Reward per Episode: -0.0019238476953907814
Convergence Speed: 499
Sample Efficiency: 6.5811623246492985
Average Path Length: 43.692
Training completed.
AutoPentest-DDPG: Optimal attack path was computed successfully
                  (labels match 'mulval_result/AttackGraph.pdf')
18->16->15->14->13->11->10->9->8->6->5->4->3->2->1
```

Figure 7. Execution and terminal output for DDPG algorithm



```
root@ANUBIS:/home/tools/AutoPentest-DRL# python3 AutoPentest-AEDDPG.py logical_attack
################################################################################
AutoPentest-AEDDPG: Automated Penetration Testing Using Deep Reinforcement Learning
################################################################################
AutoPentest-AEDDPG: Operation mode:  Attack on logical network
AutoPentest-AEDDPG: Target topology: MulVAL_P/logical_topology_1.P
--------------------------------------------------------------------------------
AutoPentest-AEDDPG: Compute attack path for logical network...
Generate attack graph using MulVAL...
Process attack graph into attack matrix...
Attack matrix saved in 'AEDDPG/processdata/all_paths.npy'
Conduct AEDDPG training for up to 100 episodes...

Training Completed.
Average Reward per Episode: 8.909090909090908
Training Time: 0.08 seconds
Convergence Speed: 22
Sample Efficiency: 4.545454545454546
Average Path Length: 54.86363636363637
AutoPentest-AEDDPG: Optimal attack path was computed successfully
                    (labels match 'mulval_result/AttackGraph.pdf')
18->16->15->14->13->11->10->9->8->6->5->4->3->2->1
root@ANUBIS:/home/tools/AutoPentest-DRL# |
```

Figure 8. Execution and terminal output for AE-DDPG algorithm

4.3.1. Epsilon delay

An epsilon delay graph (ε-delay graph) Figure 9 can provide insights into the exploration-exploitation trade-off during the training process. A log-like decreasing ε-delay graph of DQN and DDPG indicates that the RL model might be prioritizing exploitation over exploration too heavily. A straight-line downward trend in the ε-delay graph suggests the RL model is prioritizing exploitation over exploration too heavily. Overall, the straight-line downward graph suggests that the AE-DDPG model suffers from severe over-exploitation, potentially missing better attack paths. While log-like graphs for DQN and DDPG models indicate a potential exploration-exploitation imbalance, they might have explored more than AE-DDPG, which makes them potentially better candidates for finding vulnerabilities due to some level of exploration. It is difficult to definitively say which model is the best performer without considering other factors alongside the ε-delay graphs.

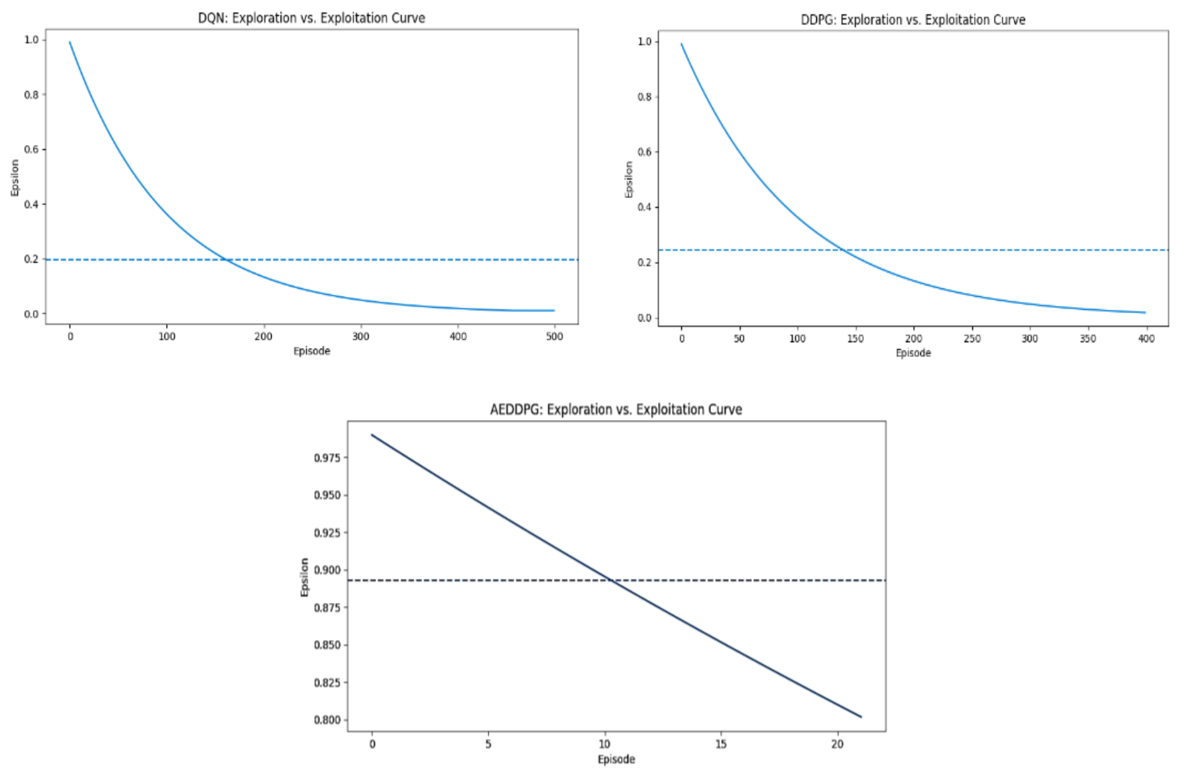


Figure 9. Exploration vs exploitation curve for DQN, DDPG, and AE-DDPG

4.3.2. Average path length

An APL graph in the context of RL models performing penetration testing provides insights into the efficiency of the model in finding vulnerabilities within the attack graph. It is the number of steps the model takes per episode, in other words, it is the number of actions of the agent in RL. It does not depend on the length of the path that contains vulnerabilities, which we refer to as the 'vulnerable path'.

A model that consistently demonstrates a lower APL throughout training episodes hints at its superior efficiency in navigating the attack graph. Such a model requires fewer steps (actions) on average to detect and exploit vulnerabilities, indicating a more direct and effective attack strategy. Conversely, a model with a consistently higher APL may be less efficient, as it necessitates more steps (actions) on average to reach vulnerabilities.

The graph Figure 10, with their distinctive peaks and troughs, provide a visual depiction of the path length for each training episode. The peaks signify episodes where the model took a longer path to identify a vulnerability, while the troughs represent episodes with shorter paths. The dotted line in the graph represents the APL. From these graphs, we can deduce that DQN boasts the lowest APL, approximately 9.610, indicating its superior efficiency in navigating the attack graph. On the other hand, AE-DDPG exhibits the highest APL of 54.864, suggesting it is the least efficient model, followed by DDPG with an APL of 34.735.

4.3.3. Average reward

The average reward per episode graph in the context of RL models performing penetration testing provides insights into the effectiveness of the model in identifying and prioritizing vulnerabilities. A model with a consistently higher average reward across training episodes suggests it is more effective at achieving the objective of the penetration testing scenario. This means the model is successfully identifying and exploiting vulnerabilities that lead to higher rewards. A model with a consistently lower average reward might be less effective.

The graph Figure 11 with total reward on the Y-axis and episode on the X-axis shows the cumulative reward achieved by the RL model throughout training. However, it does not directly display the average reward per episode. In this experiment, we meticulously calculated the average reward from this graph by dividing the X-axis (number of episodes) into equal intervals. For each interval, we estimated the average slope of the cumulative reward line, a process that requires careful attention to detail. A steeper slope indicates a higher average reward for episodes within that interval. After calculating an estimated average reward for each interval, we sum it and divide it by the total number of intervals to calculate the overall average reward, ensuring the precision and accuracy of our analysis.

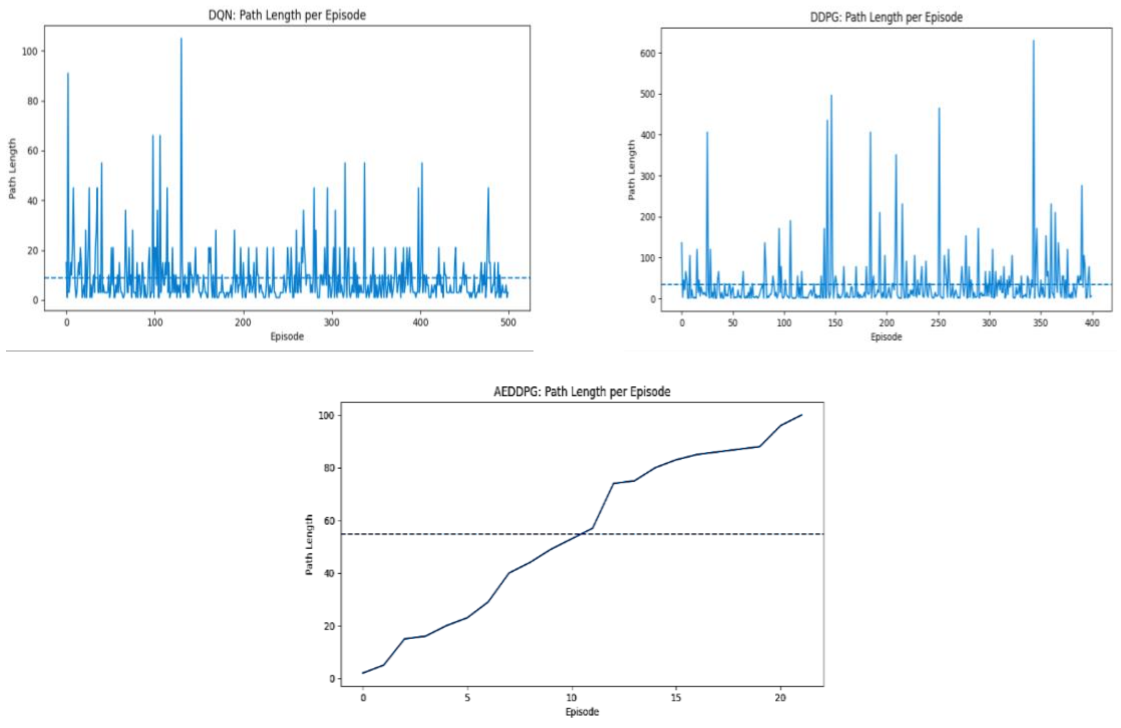


Figure 10. APL per episode for DQN, DDPG, and AE-DDPG

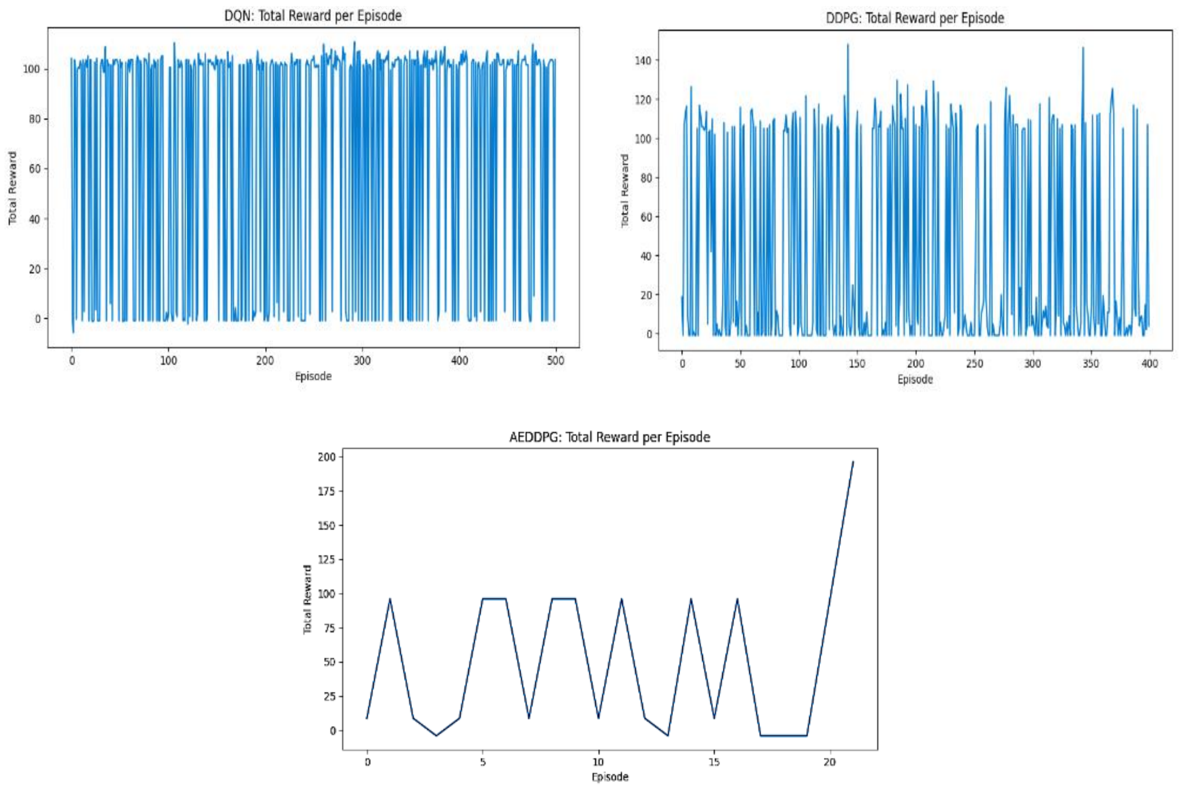


Figure 11. Average reward per episode for DQN, DDPG, and AE-DDPG

Therefore, based on the comprehensive analysis of the above graphs, it is clear that AE-DDPG is the most effective at successfully identifying and exploiting vulnerabilities. This is evidenced by its highest average reward per episode of 8.909. It is followed by DQN with 0.208, and DDPG, which is the least effective with a value of 0.010. These rankings provide a clear and certain understanding of the performance of these models in the context of penetration testing.

4.4. Inference

We have evaluated the performance of the algorithms using several metrics. The average reward per episode reflects how well the algorithm achieves its goals. Training time measures training speed, with shorter times being preferable. Convergence speed refers to how quickly the algorithm learns a stable policy, with lower numbers indicating faster learning. Sample efficiency measures how well the algorithm uses training data, and APL reflects the number of steps the algorithm takes to reach a solution. Table 1, which summarizes the experimental results from which we can infer that AE-DDPG demonstrates superior performance in comparison to DDPG and DQN.

Based on these results, it is evident that AE-DDPG demonstrates the best performance in terms of average reward per episode, training time and convergence speed. We can also infer that the path lengths of the model-free algorithms are significantly higher than the model-based algorithm. Due to the fact that more than one neural network is used in the model-free algorithms, the complexity of steps taken, and path length is increased.

Table 1. Experimental results of the algorithms in terms of the performance metrics

Performance metric	AE-DDPG	DDPG	DQN
Average reward per episode	8.909	0.010	0.208
Training time (in seconds)	0.08	8.92	3.80
Convergence speed (in episodes)	22	399	499
Sample efficiency	4.545	5.820	3.257
APL	54.864	34.735	9.610

5. CONCLUSION

In this paper, we proposed an automated penetration testing framework that leverages RL to identify and prioritize vulnerabilities within a network. The framework was evaluated using three RL algorithms-DQN, DDPG and AE-DDPG-to compare their effectiveness in automated vulnerability assessment. Our experimental results indicate that while model-based RL algorithm DQN demonstrated shorter APL compared to model-free algorithms DDPG and AE-DDPG, the latter two algorithms showed competitive performance in other metrics such as average reward per episode and faster convergence speed. Overall, our framework shows promise in automating and enhancing the penetration testing process, leading to more efficient and effective security assessments. Automated penetration testing with RL has the potential to minimize the time and resources necessary for security assessments, helping firms uncover and address vulnerabilities faster, strengthening the overall security posture, and enhancing the resilience of systems against emerging cyber threats. Further research can delve into optimizing the existing RL algorithms used within the framework. This could involve hyperparameter tuning, exploring more advanced RL architectures, and potentially incorporating multi-objective reward functions to incentivize not only shorter paths but also the discovery of more critical vulnerabilities. Scaling the framework to handle larger and more intricate network environments is crucial for real-world applications.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Suresh Jaganathan	✓	✓		✓	✓	✓	✓			✓		✓	✓	
Mrithula Kesavan Latha		✓	✓	✓		✓		✓	✓		✓			
Krithika Dharanikota		✓	✓	✓		✓		✓	✓		✓			

C	: **C**onceptualization	I	: **I**nvestigation	Vi	: **Vi**sualization
M	: **M**ethodology	R	: **R**esources	Su	: **Su**pervision
So	: **So**ftware	D	: **D**ata Curation	P	: **P**roject administration
Va	: **Va**lidation	O	: Writing - **O**riginal Draft	Fu	: **Fu**nding acquisition
Fo	: **Fo**rmal analysis	E	: Writing - Review & **E**diting		

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

Data availability is not applicable to this paper as no new data were created or analyzed in this study.

REFERENCES

[1] M. Alhamed and M. M. H. Rahman, "A systematic literature review on penetration testing in networks: future research directions," *Applied Sciences*, vol. 13, no. 12, Jun. 2023, doi: 10.3390/app13126986.

[2] F. A.-Dabaseh and E. Alshammari, "Automated penetration testing: an overview," in *Computer Science & Information Technology*, Apr. 2018, pp. 121–129, doi: 10.5121/csit.2018.80610.

[3] M. S. Barik, A. Sengupta, and C. Mazumdar, "Attack graph generation and analysis techniques," *Defence Science Journal*, vol. 66, no. 6, Oct. 2016, doi: 10.14429/dsj.66.10795.

[4] O. Almazrouei and P. Magalingam, "The internet of things network penetration testing model using attack graph analysis," in *2022 International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, Oct. 2022, pp. 360–368, doi: 10.1109/ISMSIT56059.2022.9932758.

[5] B. Yuan, Z. Pan, F. Shi, and Z. Li, "An attack path generation methods based on graph database," in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference*, Jun. 2020, pp. 1905–1910, doi: 10.1109/ITNEC48623.2020.9085039.

[6] M. Sewak, "Mathematical and algorithmic understanding of reinforcement learning," in *Deep Reinforcement Learning*, Singapore: Springer Singapore, 2019, pp. 19–27, doi: 10.1007/978-981-13-8285-7_2.

[7] C. Sarraute, O. Buffet, and J. Hoffmann, "Penetration testing== POMDP solving?," *IJCAI Workshop on Intelligent Security*, pp. 1–3, Jun. 2013.

[8] M. C. Ghanem, T. M. Chen, and E. G. Nepomuceno, "Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks," *Journal of Intelligent Information Systems*, vol. 60, no. 2, pp. 281–303, Apr. 2023, doi: 10.1007/s10844-022-00738-0.

[9] R. Gangupantulu *et al.*, "Using cyber terrain in reinforcement learning for penetration testing," in *2022 IEEE International Conference on Omni-layer Intelligent Systems*, Aug. 2022, pp. 1–8, doi: 10.1109/COINS54846.2022.9855011.

[10] A. M. Hafiz, "A survey of deep Q-networks used for reinforcement learning: state of the art," in *Intelligent Communication Technologies and Virtual Mobile Networks*, Singapore: Springer, 2023, pp. 393–402, doi: 10.1007/978-981-19-1844-5_30.

[11] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv-Computer Science*, pp. 1–13, Jul. 2015.

[12] Z. Zhang, J. Chen, Z. Chen, and W. Li, "Asynchronous episodic deep deterministic policy gradient: toward continuous control in computationally complex environments," *IEEE Transactions on Cybernetics*, vol. 51, no. 2, pp. 604–613, Feb. 2021, doi: 10.1109/TCYB.2019.2939174.

[13] W. Du and S. Ding, "A survey on multi-agent deep reinforcement learning: from the perspective of challenges and applications," *Artificial Intelligence Review*, vol. 54, no. 5, pp. 3215–3238, 2021, doi: 10.1007/s10462-020-09938-y.

[14] W. Chen, "Open problems and modern solutions for deep reinforcement learning," *arXiv-Computer Science*, pp. 1–13, Feb. 2023.

[15] J. Yi and X. Liu, "Deep reinforcement learning for intelligent penetration testing path design," *Applied Sciences*, vol. 13, no. 16, Aug. 2023, doi: 10.3390/app13169467.

[16] D. Tayouri, N. Baum, A. Shabtai, and R. Puzis, "A survey of MulVAL extensions and their attack scenarios coverage," *IEEE Access*, vol. 11, pp. 27974–27991, 2023, doi: 10.1109/ACCESS.2023.3257721.

[17] L. V. Hoang, N. X. Nhu, T. T. Nghia, N. H. Quyen, V.-H. Pham, and P. T. Duy, "Leveraging deep reinforcement learning for automating penetration testing in reconnaissance and exploitation phase," in *2022 RIVF International Conference on Computing and Communication Technologies*, Dec. 2022, pp. 41–46, doi: 10.1109/RIVF55975.2022.10013801.

[18] S. Raj and N. K. Walia, "A study on metasploit framework: a pen-testing tool," in *2020 International Conference on Computational Performance Evaluation*, Jul. 2020, pp. 296–302, doi: 10.1109/ComPE49325.2020.9200028.

[19] F. M. Zennaro and L. Erdődi, "Modelling penetration testing with reinforcement learning using capture-the-flag challenges: trade-offs between model-free learning and a priori knowledge," *IET Information Security*, vol. 17, no. 3, pp. 441–457, May 2023, doi: 10.1049/ise2.12107.

[20] S. Zhang, D. Caragea, and X. Ou, "An empirical study on using the national vulnerability database to predict software vulnerabilities," in *Database and Expert Systems Applications*, 2011, pp. 217–231, doi: 10.1007/978-3-642-23088-2_15.

[21] P. Swazinna, S. Udluft, D. Hein, and T. Runkler, "Comparing model-free and model-based algorithms for offline reinforcement learning," *IFAC-PapersOnLine*, vol. 55, no. 15, pp. 19–26, 2022, doi: 10.1016/j.ifacol.2022.07.602.

[22] X. Ou, S. Govindavajhala, and A. W. Appel, "MulVAL: a logic-based network security analyzer," *14th USENIX Security Symposium*, pp. 113–128, 2005.

[23] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *arXiv-Computer Science*, pp. 1–9, 2013.

[24] O. Valea and C. Oprişa, "Towards pentesting automation using the metasploit framework," in *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing*, 2020, pp. 171–178, doi: 10.1109/ICCP51029.2020.9266234.

[25] K. T. Banu and M. Deepthi, "Detecting, analyzing, and evaluation of vulnerabilities using metasploitable," in *Proceedings of the 2nd International Conference on Cognitive and Intelligent Computing (ICCIC 2022)*, Springer, 2023, pp. 167–175, doi: 10.1007/978-981-99-2742-5_18.

BIOGRAPHIES OF AUTHORS

Suresh Jaganathan Associate Professor in the Department of Computer Science and Engineering, Sri Sivasubramaniya Nadar College of Engineering, has more than 28 years of teaching experience. He received his Ph.D. in Computer Science from Jawaharlal Nehru Technological University, Hyderabad, M.E. Software Engineering from Anna University and B.E. in Computer Science and Engineering from Madurai Kamarajar University, Madurai. He has more than 30 publications in referred international journals and conferences. Apart from this to his credit, he has two patents in Image Processing and has written a book on "Cloud Computing: A Practical Approach for Learning and Implementation", published by Pearson Publications. He is an active reviewer in reputed journals (Elsevier - Journal of Network and Computer Applications, Computers in Biology and Medicine) and co-investigator for the SSN-NIVIDA GPU Education Centre. His areas of interest are distributed computing, deep learning, data analytics, machine learning, and blockchain technology. He can be contacted at email: sureshj@ssn.edu.in.

Mrithula Kesavan Latha is currently working as a Software Development Analyst, Citicorp Services India Ltd, Chennai. She obtained her Bachelor of Computer Science and Engineering degree from Sri Sivasubramaniya Nadar College of Engineering and a Bachelor of Science degree focused on Data Science and its Applications from Indian Institute of Technology, Madras. She has also completed a Genomic Data Science Specialization from Johns Hopkins University. Her research areas of interest include reinforcement learning, deep learning, data science, cybersecurity, artificial intelligence in healthcare and computational biology. She can be contacted at email: mrithula2010075@ssn.edu.in.

Krithika Dharanikota a Computer Science undergraduate from Sri Sivasubramaniya Nadar College of Engineering, embarking on the next chapter of her academic journey as a Master's student in Computer Science at the University of Southern California. Her passion for Computer Science is multifaceted, encompassing various exciting fields like cybersecurity, machine learning, artificial intelligence, and natural language processing (NLP). She is eager to learn and contribute to the ever-evolving field of computer science. She can be contacted at email: dharanik@usc.edu.