Automated DoS Penetration Testing Using Quantile Regression and Deep Q-Learning Network Algorithms

Mariam Alhamed, M M Hafizur Rahman
Department of Computer Networks and Communications-CCSIT, King Faisal University, Al-Ahsa, 31982, Saudi Arabia

***Abstract*—Penetration test is essential to determine the security level of a network. A penetration test attack path simulates an attack to identify vulnerabilities, reduce likely losses, and continuously enhance security. It helps to facilitate the simulation of different attack scenarios, develops robust security measures, and enables proactive risk assessment. We have combined MulVAL with DQN and QR-DQN algorithms to solve the problem of incorrect route prediction and problematic convergence associated with attack path planning training. As a result of this algorithm, an attack tree is generated, paths within the attack graph are searched for, and a deep-first search method is used to create a transfer matrix. In addition, QR-DQN and DQN algorithms determine the optimal attack path for the target system. The results of this study show that although the QR-DQN algorithm requires more resources and takes longer to train than the traditional (DQN) algorithm, it is effective in identifying vulnerabilities and optimizing attack paths.**

Keywords*—*DQN; QR-DQN; MulVAL; DFS; penetration testing; DoS

I. Introduction

Recently, network security has been considered a critical issue that needs to be addressed. Networks connected to the internet are inherently insecure and can be abused by hackers, regardless of whether they are wired or wireless. When transmitted, data passes through numerous terminals before reaching its destination, allowing corrupt users to intercept or modify it.

Due to the increasingly complex and aggressive threats to network security, the researchers explained that an effective strategy to tackle this problem is to investigate the aspects of network security of a system through penetration testing. Penetration testing is an essential approach to determine the security level of a network system. Penetration testing involves simulating an attack in multiple attack scenarios to ensure the security of the system or environment under investigation. We can reduce possible risks by eliminating these vulnerabilities in advance and increasing the system's security. However, penetration testing can be performed manually or automatically.

Manual penetration tests require exceptional skills. Automated penetration testing has recently gained popularity as a "hot spot" in network security. Planning the attack path is an important phase of automated penetration testing. Thorough planning is essential in automated penetration testing, ensuring that the attack path is well-defined and comprehensive. By carefully planning the path of the attack, organizations can effectively model real-world attack scenarios and recognize possible vulnerabilities in their systems and networks. This helps uncover hidden security vulnerabilities and enables targeted remediation measures to strengthen security posture. By uncovering such vulnerabilities and understanding the potential impact of a real-world attack, companies can take proactive measures to strengthen their security defenses and protect themselves against similar threats in the future. Several sophisticated AutoPT methods and frameworks have been created to improve penetration test performance through reinforcement learning RL or deep reinforcement learning. Both Reinforcement learning (RL) and deep reinforcement learning (DRL) have shown promise in improving penetration test performance. DRL is better than RL because it uses deep neural networks to handle complicated, high-dimensional data, enabling more accurate and efficient vulnerability discovery. Furthermore, DRL can learn directly from raw data, eliminating the requirement for feature engineering and reducing manual intervention. As a result, DRL-based automated penetration testing systems have the potential to provide more comprehensive and reliable security assessments. DRL algorithms can learn and adapt to different network environments, allowing them to navigate complex systems and detect vulnerabilities more efficiently [7].

The DRL approach differs from typical machine learning approaches that use labeled data for supervised learning instead of learning optimal tactics through interaction with the environment. This enables the agent to learn through trial and error and constantly improves its methods based on the rewards it receives. In addition, DRL can deal with more complex and dynamic environments where predefined labels and models are inadequate. DRL algorithms are classified into three main categories: strategy-based search, model-based methods, and value-based functions. In the strategy-based search, the agent focuses on learning a strategy that maps states directly to actions.

The main contribution of this study is as follows:

1) First application of QR-DQN in automated penetration testing: This study is one of the first to use the Quantile Regression Deep Q Network (QR-DQN) for automated penetration testing. Although previous research has focused predominantly on classical DQN and its variants (e.g. double DQN, dueling DQN) [15] [19], our study introduces the QR-DQN model, which estimates the distribution of rewards and not just the expected values. This provides a more robust framework for decision-making under uncertainty and low incentives, a key challenge in penetration testing.

2) Closing the vulnerability detection gap: This study fills a critical literature gap by offering a novel method for more effective vulnerability detection in complex network environments. Traditional penetration testing often provides limited results due to difficulty in identifying exploitable vulnerabilities. Our approach with QR-DQN improves the exploration of attack surfaces and makes the detection of vulnerabilities more efficient in networks where exploitable paths are difficult to find.

3) Improved reconnaissance and discovery of hidden paths: Using the QR-DQN distribution approach, this study significantly improves reconnaissance capabilities. QR-DQN enables the model to capture a broader range of possible outcomes, allowing the agent to discover hidden attack paths and vulnerabilities that classical DQN-based models may miss [15]. This makes the model better able to deal with inherent uncertainty in penetration testing, where attack success and vulnerability exploitation are often unpredictable [19]. Quantile-based assessment allows the agent to make more risk-aware and adaptive decisions, significantly improving models based on expected gains.

4) Comprehensive empirical comparison: This study compares QR-DQN and traditional DQN models in simulated penetration testing environments.

These contributions show that QR-DQN has the potential to revolutionize automated penetration testing by improving decision-making and reconnaissance and providing better adaptability to different network environments.

II. Related Works

The study by Hu Z, Beuran R, and Tan Y [1] aimed to Automate penetration testing as part of cybersecurity training by incorporating Deep Reinforcement Learning. This methodology leads to directed learning for attack training, suggesting potential techniques. The authors conducted automated penetration testing in two phases. The security tools used were the Shodan search engine, MulVAL, and the DQN method. Finally, they found that the framework is useful to suggest attack strategies.

Maeda R and Mimura M [2] aimed to study the behavior of the attackers to assess the risks after a successful explosion. They, therefore, proposed to automate post-exploitation using reinforcement learning. They combined deep reinforcement learning with PowerShell Empire. In addition, they proposed three reinforcement learning models and then conducted two phases to develop the models: the learning phase and the testing phase. In conclusion, they found that the proposed methods are very suitable for obtaining the administrative rights for the domain controller.

Masarweh.A [3] proposes Threat Led APT PT, which is an extended PT technique that tests a target network's security for existing vulnerabilities. This study employs a variety of APT attack strategies to uncover hidden vulnerabilities. In addition, he created a new dataset by gathering traffic from actual APT assaults and tested it with a machine learning model to detect APT attacks. The author discovered that the suggested model greatly improved network security by 14 to 28.5 percent. Furthermore, the proposed model outperformed existing classifiers in terms of power and efficiency for detecting APT assaults.

Goh.KC [4] proposed automated penetration testing using reinforcement learning. The phases of penetration testing were vulnerability scanning, exploitation, and post-exploitation, but not information gathering. They use Nmap tools and then send the information to the reinforcement learning agent to make the best prediction to exploit the system. They found that reinforcement learning has the potential to increase the performance of automated penetration testing and reduce testing resources. In addition, the reinforcement learning algorithm was found to reduce time and increase the probability of exploitation during automated penetration tests. The resulting error was discovered, but a simple algorithm such as Q-learning still achieves a remarkable result.

Huizinga.T [5] developed a technique for analyzing network data using machine learning to ensure the verifiability of penetration testing. The findings of this study demonstrate that preprocessing and classification may be completed quickly enough to be conducted live during a pen test. Thus, this model was very accurate. The author recommended that a new model be created with a different classification of all traffic.

Chu. G and Lisitsa. A [6] suggested penetration testing automation, an agent-based belief-desire-intention (BDI) paradigm. They employed Agent Speak Jason, a programming language for multi-agent systems based on the Belief-Desire-Intention (BDI) paradigm. The author utilized two agents: the target agent and the BDI agent. Finally, the authors discovered that the simulation accurately depicts the BDI agent's behavior and mental process, hence validating the modeling.

Sommerville ÅÅ et al. [7] used reinforcement learning (Q) bots to simulate a SQL injection vulnerability, demonstrating white-hat hacking techniques. The authors characterized it as a Markov decision process (MDP) and used reinforcement learning. They discovered that both interpretable and basic tabular Q-learning agents, as well as more advanced deep Q-learning agents, are capable of learning useful strategies. Finally, they discovered that the taught technique is less likely to perform optimally in additional cases.

Tran. K et al. [8] used Deep Hierarchical Reinforcement Agents (HA-DRL) to automate penetration testing. Compared to a traditional Deep-Q-Learning (DQN) agent, a common technique for using artificial intelligence in automated penetration testing, they found the ideal attack strategy to be faster and more continuous. The proposed method is suitable for exploring huge action spaces.

The study by Koroniotis. N et al. [9] attempted to create methods for detecting vulnerabilities in smart IoT systems. They created a deep learning-based penetration testing system known as Long Short-Term Memory Recurrent Neural Network Enabled Vulnerability Identification (LSTM-EVI). They utilized a test environment to obtain network data. The models were taught to return zero for regular traffic and one for assaults. Finally, the models outperformed existing coercive strategies in identifying scanning assaults.

Kujanpa¨a.K et al. [10] created a computer simulation of the potential risk posed by malicious actors teaching automated bots to extend local privileges using deep reinforcement learning. They discovered that, depending on the configuration of the environment it encounters, the model can elevate privileges in a Windows 7 environment via a variety of approaches.

There are 38 actions specified in the vulnerability action space that allow privilege escalation. The model may be useful for training and testing intrusion detection systems, as the agent can generate realistic attack sensor data.

In the study by Neal. C et al. [11], the goal was to find malicious inputs that reduce the effectiveness of microgrid control. These are compact power systems that interconnect loads and a variety of dispersed power sources in specific areas. Therefore, they tested the pervasiveness of microgrid control algorithms using reinforcement learning. The MGs architecture was implemented using MATLAB/Simulink, and RL was used to teach the agent how to change the results of malicious inputs to the MGs controller. Finally, they discovered that the attacks generated showed that the overall performance of the controller was most affected by lowering the reported battery SOC.

The study of Semenov.S et al. [12] was to improve the security of computer networks, so they performed automated penetration testing based on Deep Reinforcement Learning. They used the capabilities of the Shadon system to collect real-world data for designing attack trees. Then, the Mulval platform was used to build attack trees. A method was developed to build a matrix of cyber-attacks using the Mulval tool. They used CVSS scoring to assign reward points to each node to reduce the attack tree and identify an attack with a higher probability of occurrence. They found that the model is suitable for software security analysis because it allows the auditor to choose a sound ethical hacking policy and measures to mitigate the negative factors of potential cyber-attacks.

Tran. K et al. [13] suggested an automated penetration testing technique based on Deep Machine Learning (CRLA). The complexity of the suggested cybersecurity network develops exponentially, and this approach sought to decrease discrete action spaces in an autonomous penetration test model. In large-scale action space situations, they observed that the model's optimal attack policy is quicker and more stable than a conventional deep-Q learning agent.

Zennaro. F and Erdodi. L [14] ran models by using RL to solve the basic penetration testing problem in the form of capture-the-flag hacking challenges. They used three classes of CTF problems to build the models, which are port scanning and intrusion, server hacking, and website hacking, and they analyzed how model-free reinforcement learning algorithms can help solve these problems. It is critical to provide agents with prior knowledge in order to achieve effective solutions.

Zhou et al. [15] treated model penetration testing as a Markov Decision Process (MDP) problem and employed reinforcement learning technology to do autonomous penetration testing in huge networks. The suggested model, NDSPIDQN, seeks to address two issues in large-scale scenarios: the sparse reward problem and the huge action space problem. They utilized five DQN extensions. They then separated the action and divided the neural network estimators to calculate two aspects of the action independently. The experiment employs PyTorch as the algorithm framework and takes place in the following experimental environment: NVIDIA Geforce RTX3090 GPU, Intel Xeon Gold 6248R CPU, and 64GB RAM. Finally, they evaluated a variety of scenarios with the algorithms. They discovered that the techniques had superior convergence and scaling performance.

Gangupantulu. R. et al. [16] provided strategies for building attack graphs on the cyber battlefield using concepts from IPB. They considered a motivating case where firewalls are viewed as obstacles and are reflected in both the state dynamics and the reward space. They have shown how to realistically design attack graphs for RL using terrain analysis. To demonstrate the concept, the authors used an attack graph with about 1000 nodes and 2300 edges and Deep Q reinforcement learning with experience replay.

Chowdary.A. et al. [17] suggested a framework for automated penetration testing to solve the problem of large-scale penetration testing. They used attack graphs to generate a map of security threats and probable attack vectors across the network. In addition, they used reinforcement learning based on a Deep-Q Network (DQN) to determine the best penetration testing strategy, as well as a domain-specific transition matrix and reward modeling to capture the significance of security vulnerabilities and the challenges of exploiting them.

Zhang. Y et al. [18] proposed modeling the black box penetration testing procedure as a Certainly noticed Markov Decision procedure (POMDP) to characterize the transitions in a real-world scenario. They also presented a new method, ND3RQN, for automated black box penetration testing. They also employed a Long Short-Term Memory (LSTM) framework, which allows the agent to make judgments based on past memories. They employed a neural network structure. They discovered that the unique algorithm can generate a bigger attack route approach for all susceptible hosts during automated black box penetration tests.

Yang. Y et al. [19] attempted autonomous penetration testing in the framework of Multi-Objective Reinforcement Learning (MORL) and suggested a crucial Chebyshev decomposition to identify alternative adversarial strategies that balance diverse penetration test objectives. To assist the agent in adjusting to future excursions, scientists included a coverage-based masking technique that gives less weight to previously selected actions. According to their findings, the suggested technique outperforms modified algorithms in terms of multi-centric learning and performance efficiency.

On the basis of the studies discussed, we have established the following:

- Sparse rewards and large action spaces are common challenges in many studies. Several works, such as [15] and [16], focused on solving these problems by proposing advanced reinforcement learning models, such as NDSPIDQN and Deep Q-learning with experience repetition, but reward uncertainty remains a key challenge.
- The study of complex networks has been a major focus in studies such as [8] and [19]. These studies have shown that hierarchical reinforcement learning and multi-criteria reinforcement learning can improve exploration in large action spaces, although further work is needed to refine exploration strategies for complex environments.
- Post-exploitation and APT recognition have been explored in papers such as [2] and [3]. These studies

utilized reinforcement learning to simulate the post-exploitation phase, focusing in particular on privilege escalation. However, these approaches lack the ability to fully model reward uncertainty during the post-exploitation phase, limiting their long-term planning capabilities.

- In terms of environments, many studies have applied reinforcement learning to IoT networks or other complex systems such as [9] and [4]. Although IoT brings unique challenges, most models still struggle with real-time adaptability and scalability, especially in dynamic network environments.
- Reward shaping and mitigation of sparse rewards were key techniques in some studies such as [12] and [15]. While CVSS-based reward shaping helped to reduce sparse rewards, the models still reached their limits when applied to large and complex attack spaces.
- Several studies integrated real-world tools such as Shodan and MulVAL for real-world penetration testing such as [1] and [12], demonstrating practical applications of reinforcement learning in security testing. However, further optimization is needed to cope with the variability of rewards in these dynamic scenarios.
- The automation of penetration testing has been improved in studies such as [6]and [17] by using reinforcement learning to automate the detection of attack paths. However, most of these models lack advanced techniques for dealing with uncertain rewards, leading to suboptimal decisions.

III. System Architecture for Automated DoS Penetration Testing

DRL algorithms are used to create an automated penetration testing system for Denial of Service (DoS) attack scenarios. The primary goal is to identify optimal attack paths within the network. For this purpose, we use value-based deep reinforcement learning methods such as Deep Q-learning networks (DQN) and Quantile Regression Deep Networks (QR-DQN). While DQN is effective in generally stable and low-risk contexts, QR-DQN offers significant advantages when dealing with the complexities and uncertainties associated with automated penetration testing.

This study compares the performance of DQN and QR-DQN in identifying optimal attack paths in a DoS penetration testing scenario. The focus is on evaluating which model performs better in terms of vulnerability detection, path optimization, and efficiency in large networks. The study includes several steps to train and evaluate the system. We simulate network environments and create two scenarios using the same vulnerability dataset for training and testing.

The vulnerability data comes from the National Vulnerability Database (NVD), which contains up-to-date information on current vulnerabilities. This dataset ensures that the training scenarios are realistic and reflect current security threats. The host dataset describes network topologies, while the vulnerability dataset contains critical vulnerabilities. By including the latest vulnerabilities from the NVD, the system can accurately assess and respond to potential security issues. Each simulated network is populated with hosts and assigned vulnerabilities from the NVD dataset, representing a realistic threat environment. These networks are then processed using the MulVAL tool, which generates attack trees that visualize the potential attack paths that attackers could use to compromise network systems. Once the attack tree has been generated, the next step is to convert it into a matrix that can be used as input for the DQN and QR-DQN models. To do this, we use the Depth-First Search (DFS) algorithm. It was chosen for its ability to thoroughly investigate all possible attack paths from the root to the target nodes. This ensures that every potential attack path is considered, even in large and complex networks.

The DFS algorithm traverses the attack graph generated by MulVAL. It converts it into a structured matrix in which each row represents an attack path, and each column represents characteristics such as CVE IDs, exploitability scores, and other network attributes. Once the DFS algorithm has simplified the attack tree into this matrix format, it is used as input to the DQN and QR-DQN models, which are trained to identify the optimal paths to exploit vulnerabilities. Once the data has been pre-processed, the DQN and QR-DQN models begin training.

DQN learns by representing each network state as a node in the attack path and selecting the best action (exploiting a vulnerability or moving along a path) based on the expected reward. At the same time, QR-DQN extends this approach by estimating the overall distribution of future rewards. This makes it more suitable for highly uncertain environments, such as penetration testing scenarios with sparse or uncertain rewards.

The models are assessed using Common Vulnerability Scoring System (CVSS) scores to determine their efficiency in detecting critical vulnerabilities, reducing false negatives, and optimizing attack paths. The reward function is important in this evaluation since it guides the learning process using static CVE impact values as well as adaptive incentive methods. CVE impact scores serve as baseline incentives, ensuring that vulnerabilities of greater severity and exploitability contribute more to learning. This study compares DQN and QR-DQN to determine which model is more effective in identifying optimal attack paths in DoS penetration testing. It aims to improve the overall efficiency and accuracy of automated security assessments (Fig. 1).

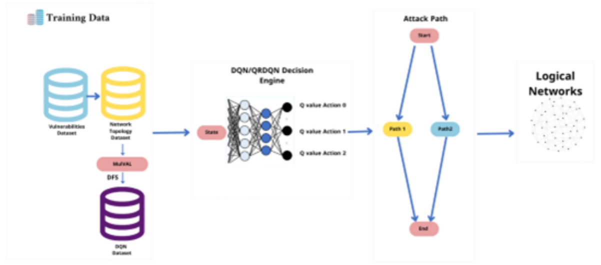


Fig. 1. System architecture for automated DoS penetration testing.

IV. Proposed Method Components

The aim of this study is to test vulnerabilities against DoS attacks using an automated penetration testing framework based on DQN and QR-DQN algorithms. The setup includes a controlled network environment to evaluate the performance of these models.

A. Hardware Components

The hardware equipment required to simulate the DoS test environment includes:

1) Router (5G capable): Serves as the primary network device that is tested for DoS vulnerabilities.

2) Computing device: A computer equipped with an AMD Ryzen 7 processor and 16 GB of RAM connected to the internet to support testing and simulation.

B. Software Components

The software was developed to create virtual networks, implement algorithms, and execute the automated test procedure:

1) VirtualBox: Used to set up virtual environments that allow the simulation of different network topologies and multiple hosts in a controlled environment.

2) Ubuntu 24.0: Used in the virtual machine environment and provides a stable platform for running simulations and test processes.

3) Python 3.11 and PyTorch: Python serves as the primary programming environment, with PyTorch supporting the implementation of the DQN and QR-DQN algorithms for training and evaluating the models for DoS penetration testing.

These components were used to perform tests in various network scenarios and evaluate the DQN and QR-DQN models using metrics such as accuracy, speed, and adaptability in automated DoS penetration testing.

V. Penetration Testing in Deep Reinforcement Learning

Using DRL algorithms, penetration testers simulate and optimize attack strategies in networks and attempt to identify vulnerabilities in an automated, adaptive, and effective manner. Unlike traditional penetration tests, which are based on predefined attack patterns and manual processes, DRL-based penetration tests enable dynamic exploration and adaptation, allowing test agents to independently discover new attack paths and strategies based on feedback from the environment.

Based on the unique requirements of automated DoS penetration testing, we discussed the different models in DRL and explained why DQN and QR-DQN were selected as the most effective options. They are as follows:

A. Policy-based Models (A3C and PPO)

Policy-based approaches, such as Asynchronous Advantage Actor-Critic (A3C) and Proximal Policy Optimization (PPO), are successful in dealing with continuous action spaces and have proven to be robust in real-time decision applications. However, these models often require large computational resources and accurate adaptation to balance exploration and exploitation, making them less adaptable for dynamic contexts such as penetration testing. Smith, J., and Lee, A. (2022) state in their paper that while A3C and PPO are effective in continuous action spaces, they can be inefficient in discrete cybersecurity scenarios.

B. Hierarchical Models (HA-DRL)

Hierarchical models, such as Hierarchical Actor-Critic (HA-DRL), are designed to enable multi-level decision-making, which can be useful when dealing with complicated tasks. However, they are often computationally intensive and difficult to implement, especially for applications that require fast, simplified decision-making, such as penetration testing. Chen L. et al. (2023) show in their work on network intrusion detection that hierarchical models are successful but require a huge amount of computation and sophisticated configuration. This makes them unsuitable for real-time network security applications, as DQN can provide more efficient performance with less complexity (Proceedings of the International Conference on Network Security).

C. Quantile Regression Deep Q-Network (QR-DQN)

It can represent reward distributions, including the diversity and uncertainty associated with penetration testing. This property makes QR-DQN suitable for use in dynamic situations with unexpected attack paths and rewards. Li, X., and Zhao, Y. (2021) used QR-DQN for intrusion detection and demonstrated its robustness in insecure environments, which fits well with the requirements of penetration testing where attack success and exploitability can vary greatly (Proceedings of the ACM Workshop on Artificial Intelligence and Security).

D. Deep Q-Network (DQN)

DQN provides a solid foundation for detecting attack vectors in penetration tests. Due to its simple architecture and efficiency in complicated contexts, it is widely used in network security. Wang, H., et al. (2021) discuss the use of DQN in automated penetration testing and show that it is able to efficiently navigate complex network structures and identify optimal attack paths, supporting its use as a reliable model in network security (Computers & Security, 102, 102156).

VI. Selected Models

We chose to compare DQN and QR-DQN models for automated DoS penetration testing based on findings from previous research demonstrating their effectiveness in cybersecurity and automated penetration testing scenarios. Based on these studies, we can conclude that the models can handle complex environments, optimize decision-making under uncertainty, and improve the identification of vulnerabilities in network configurations.

A. Application of DQN in Penetration Testing

Several studies have demonstrated the effectiveness of DQN in automating penetration tests. For example, Hu, Beuran, and Tan (2020) used DQN to automate network vulnerability assessments and showed that DQN is well suited to attack path selection and can efficiently balance reconnaissance and exploitation in static network environments. This work shows that DQN can simplify the task of pathfinding in large, multi-layered networks by learning from past actions and optimizing its strategy over time. The effectiveness of DQN in large state spaces, as demonstrated in their study, supports its application in identifying potential DoS attack paths in complex network scenarios.

B. Advantages of QR-DQN for Dealing with Uncertainty

Traditional DQN models, while effective, are often limited when handling scenarios with sparse or uncertain rewards, such as those common in penetration testing, where successful attack paths do not always yield immediate rewards. The study by Dabney et al. (2018) on QR-DQN shows that it is able to capture the distribution of possible future rewards, providing a more robust approach to decision-making under uncertainty. The quantile-based approach of QR-DQN allows a spectrum of possible outcomes to be modeled, making it particularly useful for cybersecurity tasks where potential attack paths have different probabilities of success. This has been confirmed by research in insecure environments, where QR-DQN consistently outperformed DQN in identifying optimal solutions under risk.

C. Scalability and Adaptability in Complex Environments

Research by Goh et al. (2021) and Koroniotis et al. (2022) has emphasized the importance of scalable models, such as DQN and QR-DQN, for penetration testing in large network topologies. Their studies have shown that these models are highly adaptable, with DQN efficiently handling simple attack simulations, while QR-DQN provides superior performance in scenarios with complex network structures and high variability of reward signals. QR-DQN's ability to generalize across different environments suggests that it can adapt to changes in network configurations, making it a valuable choice for automated DoS testing where network topologies may evolve.

D. Improved Vulnerability Detection Through Distribution-Based Learning

Studies such as those by Masarweh (2021) and Zhou et al. (2023) have explored the limitations of traditional DRL models in penetration testing, especially when it comes to unknown vulnerabilities. By using the distribution-based reinforcement learning approach of QR-DQN, these studies achieved higher sensitivity in detecting hidden vulnerabilities that were missed by simpler models. QR-DQN's distribution-based approach has been shown to contribute to risk-aware decision-making, - a crucial factor in penetration testing, where the consequences of overlooked vulnerabilities can be severe. This supports QR-DQN as the optimal choice for scenarios that require a deeper understanding of potential threats.

VII. Proposed System Architecture

This section describes the architecture used to implement automated penetration testing with DQN and QR-DQN within a deep reinforcement learning framework. The framework consists of the following components:

A. Training Dataset

In this study, we use the dataset originally presented in the research titled "Automated Penetration Testing Using Deep Reinforcement Learning" by Zhenguo Hu, Razvan Beuran, and Yasuo Tan [1], modified to focus on Denial of Service (DoS) attacks. The training dataset consists of two main elements: the host dataset and the vulnerability dataset, which are used as input for the MulVAL tool to generate attack paths.

1) Host dataset: In the host dataset, we simulate two different network scenarios, each representing different network topologies with different configurations, hosts, and services. These scenarios are designed to provide different training and testing environments, using the same vulnerability dataset in both scenarios. The host configurations are represented in a logical topology format (.p file) that is input into the MulVAL tool to generate attack paths based on the specified vulnerabilities.

2) Vulnerability dataset: The vulnerability dataset is shared by both scenarios and comes from NVD dataset, with additional vulnerabilities related to DoS attacks. Each vulnerability is characterized by the following technical features:

a) CVE-ID: Unique identifier from the Common Vulnerabilities and Exposures (CVE) database.

b) Type of vulnerability: Indicates the type of vulnerability, e.g. DoS, buffer overflow, or injection.

c) Protocol: The application protocol (e.g. HTTPS, HTTP).

d) Transport layer protocol: Specifies the transport protocol (e.g. TCP, UDP).

e) Port: The port number used by the service (e.g. 443 for HTTPS).

f) Software/service: The affected software or service (e.g. Apache HTTP Server).

In addition to the features described above, a CVE info dataset is used to provide detailed information about vulnerabilities that are crucial for determining rewards during the training of the models. Each entry in the CVE info record contains the following fields:

- CVE ID: The unique identifier from the Common Vulnerabilities and Exposures (CVE) database (e.g. CVE-2023-44487).
- Vulnerability type: A description of the nature of the vulnerability (e.g. Denial of Service).
- Exploit-ability Score: A numerical value indicating the ease of exploitation (e.g. 7.5).
- Impact score: A numerical value indicating the severity of the vulnerability's impact (e.g. 10.0).

These scores are crucial for determining the reward function for the reinforcement learning models. During the training process, higher rewards are given for successfully identifying vulnerabilities with high impact and exploitability scores. This allows the models to prioritize discovering more severe vulnerabilities, leading to better optimization of attack paths.

The same vulnerability dataset is used for both the training and testing phases for both scenarios. Maintaining a consistent vulnerability dataset ensures the robustness and adaptability of the DQN and QR-DQN models when detecting vulnerabilities in different network configurations. In addition, when integrating the CVE info dataset into the reward structure, the models are guided to prioritize high-risk vulnerabilities, improving the overall effectiveness of the penetration testing framework.

B. DQN and QRDQN Decision Engine

The DQN & QR-DQN Decision Engine is a core component of the framework for automated penetration tests. It is responsible for training the models in order to identify optimal attack paths based on the attack graph generated by the MulVAL tool. In this process, the attack graph is converted into a structured matrix format using the DFS (Depth-First Search) algorithm, which the DQN and QR-DQN agents can use for training and decision-making.

1) Pre-processing with the MulVAL tool and the DFS algorithm: The MulVAL tool first generates attack trees based on the host and vulnerability data. These attack trees represent possible paths that an attacker could take to exploit vulnerabilities and reach critical targets within the network.

Before the attack paths can be fed into the DQN and QR-DQN models, the DFS algorithm is applied to the attack graph. The DFS algorithm was chosen because it is able to analyze all potential paths from the root to the leaf nodes in a sequential manner. This ensures that all possible attack scenarios are considered, even in deep and complex networks. DFS converts the attack graph into a matrix, where:

- Rows represent different attack paths.
- Columns represent characteristics of each node in the path, including information about vulnerabilities, exploitability values, and other network characteristics.

DFS is particularly suited to this task as it requires minimal memory and ensures thorough exploration, which is crucial in penetration testing, as overlooking a potential path could lead to undiscovered vulnerabilities [20].

2) DQN (Deep Q-Network): Once the attack paths are converted into a matrix, the DQN model uses this as input to train the selection of the best attack paths:

- DQN models learn by representing each state as a particular step in the attack graph (e.g. exploiting a vulnerability).
- The action represents the agent's decision to either take a particular path or exploit a particular vulnerability.
- The reward system is driven by the CVE info dataset, with higher rewards given for identifying vulnerabilities with greater severity or exploitability.
- While DQN models are effective, they are limited in their ability to deal with the uncertainty and variability of rewards, which is why QR-DQN offers additional advantages.

3) QR-DQN (Quantile Regression Deep Q-Network): The QR-DQN model also uses the matrix created by DFS but goes beyond DQN by estimating the entire distribution of future rewards and not just the expected value. This allows QR-DQN:

- Evaluate the potential range of outcomes for each action, making it better suited for environments with high uncertainty and sparse rewards, such as penetration testing.
- Make decisions based on risk distribution so that attack paths can be identified that are more promising in the long term, even if the immediate benefit is more uncertain [21].

4) Training process: After DFS has transformed the attack graph into a matrix, the data is processed:

- Both the DQN and QR-DQN models are trained to select the most effective attack paths, using rewards based on CVE data such as exploit and impact scores.
- The models are evaluated on their ability to detect critical vulnerabilities, adapt to new vulnerabilities, and optimize attack strategies while minimizing false positives and negatives.

Fig. 2 and Fig. 3 provide an overview of the architecture of the methods described in this work.

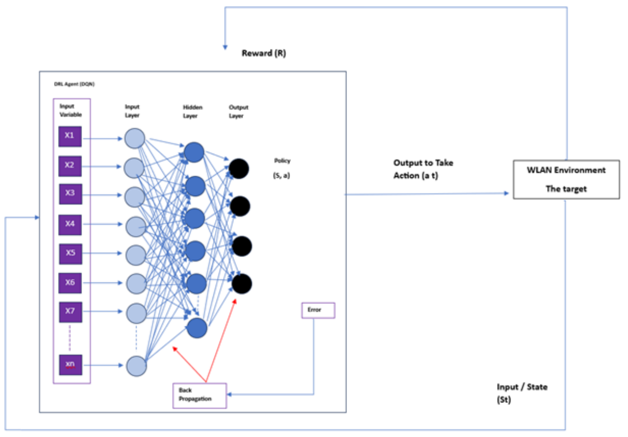


Fig. 2. Deep reinforcement learning (DQN algorithm) implementation.

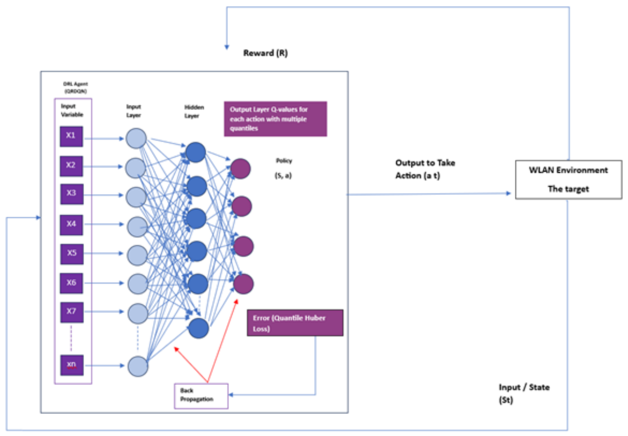


Fig. 3. Deep reinforcement learning (QR-DQN algorithm) implementation.

VIII. UML Diagram

The UML diagram (Fig. 4) provides a conceptual representation of the system architecture for the penetration tests used in this study. It illustrates the interaction of the various components, from network analysis and attack simulation to reinforcement learning agents (DQN and QR-DQN). Below is a detailed explanation of the key components and their interactions in the UML diagram:

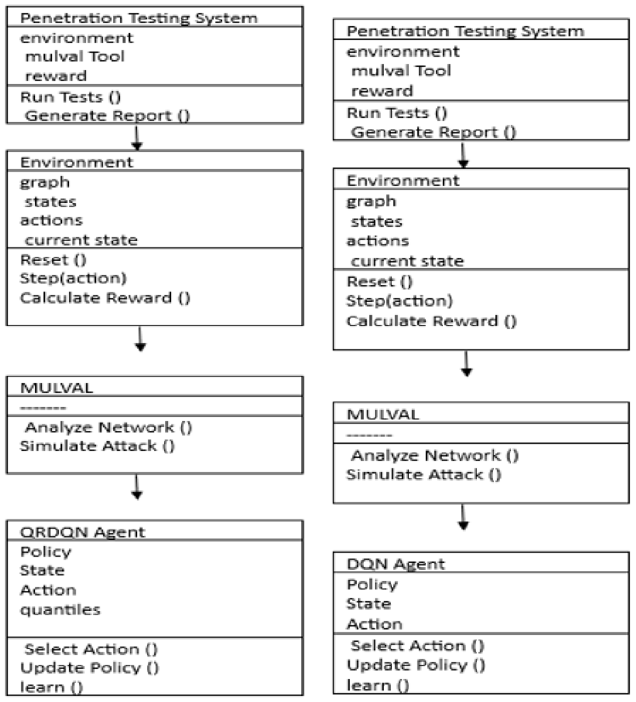


Fig. 4. UML Diagram for automated DoS penetration testing.

A. Penetration Test System (Main Controller)

At the center of the UML diagram is the penetration test system, which manages the entire automated penetration test process as the main controller. It initiates and coordinates the following processes:

1) Network configuration: It loads the network topologies and host data, assigns the vulnerabilities from the NVD dataset, and sets up the environment for the penetration test.

2) Assignment of vulnerabilities: The system assigns specific vulnerabilities from the NVD dataset to the hosts in the network for a realistic simulation.

B. Environment

The graph, the state, the action, and the current state are closely linked in the environment, which is core testing in the system. The graph defines the attack network structure, the state represents a snapshot of the network, and each agent's action changes the current state based on the success or failure of the attack. This interaction helps the DQN and QR-DQN agents learn and optimize their penetration testing strategies over time.

1) MulVAL Tool (Network analysis and attack simulation): The MulVAL tool interacts with the Penetration Test System to perform network analyses and attack simulations. Its role in the diagram includes:

a) Attack graph generation: MulVAL generates the attack graphs by analyzing the assigned vulnerabilities and network configuration. This attack graph shows potential paths that attackers could use to exploit vulnerabilities.

b) Data preparation for agents: Once the attack graph is generated, it is converted into a structured format (using the Depth-First Search (DFS) algorithm) and then converted into a matrix. This matrix serves as input for the learning agents (DQN and QR-DQN).

C. DQN Agent and QR-DQN Agent

The diagram shows the interaction between the penetration test system, MulVAL, and the DQN and QR-DQN agents. These reinforcement learning agents are responsible for learning and selecting optimal attack paths. Their roles in the UML diagram are:

1) Initialization: The penetration test system initializes both the DQN and QR-DQN agents by feeding them with the matrix generated by MulVAL. Each agent receives the same data but uses different learning techniques to optimize the selection of attack paths.

a) DQN: The DQN agent uses a value-based learning method where it learns to take the best action (choosing an attack path) based on the expected reward for exploiting vulnerabilities.

b) QR-DQN: The QR-DQN agent, on the other hand, estimates the distribution of future rewards and can thus better deal with uncertainties and improve performance in more complex scenarios.

2) Learning process: Both agents interact with the environment (represented as the matrix generated by MulVAL) to perform penetration tests:

- The agents perform actions by selecting specific vulnerabilities to exploit.

- Based on the outcome of these actions (successful or failed exploitation), the agents receive rewards based on the CVE info dataset and update their decision-making process.

3) Training and decision management: The penetration test system monitors the performance of both agents and manages the training iterations until the agents learn to select optimal attack paths. After training, the system evaluates which model (DQN or QR-DQN) performs better in identifying the most effective attack paths.

IX. Implementation Results

The tests were conducted in different network scenarios to evaluate the performance of the DQN and QR-DQN models for automated DoS penetration testing.

A. Topology

Two different network scenarios were set up for the experiments:

1) Scenario 1: A simple WLAN network in which a laptop workstation is connected wirelessly to a router while web and file servers are connected via wired connections. This configuration reflects typical environments where user devices use Wi-Fi to access network services hosted on wired servers. The topology consists of one subnet and three hosts, which is shown in Fig. 5.

2) Scenario 2: A hybrid Wi-Fi system with two subnets and three hosts. The workstation connects wirelessly and VLANs (managed by a switch) are used to segment the network. This setup adds complexity by simulating enterprise environments with segmented networks for increased security, as shown in Fig. 5. The implementation scenarios list is shown below in Table I.

TABLE I. Implementation Scenarios List

Scenario	Subnets	Hosts	Vulnerabilities Number
Scenario 1	1	3	2
Scenario 2	2	3	3

The two scenarios provided distinct environments for evaluating how well the algorithms performed under different levels of network complexity.

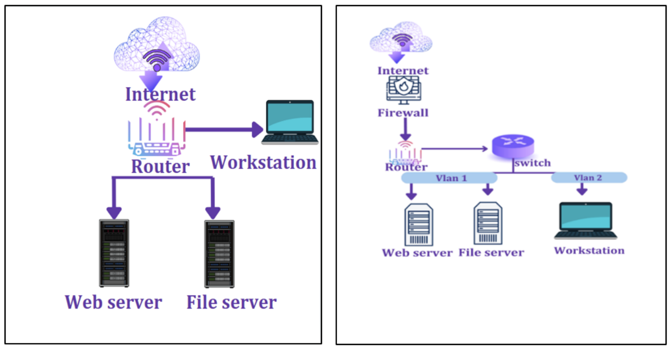


Fig. 5. Network environments for penetration testing.

B. MulVAL Tool

The MulVAL tool was used to create attack graphs for both scenarios. MulVAL interprets the network topology and configurations to create an attack graph that maps possible attack paths based on the vulnerabilities present. Each attack graph represents nodes (states or conditions) and edges (attack transitions) and illustrates how an attacker could move through the network to exploit vulnerabilities.

1) In scenario 1: The attack diagram generated focused on a DoS vulnerability (CVE-2023-44487) in the web server running Apache HTTP on port 80. Fig. 6 shows a series of steps that an attacker could take to launch a DoS attack.

2) In scenario 2: The diagram shows vulnerabilities related to services running on ports 443 (HTTPS) and 80 (HTTP), specifically vulnerabilities CVE-2021-4487 and CVE-2018-1234, which can be exploited for DoS attacks. Fig. 7 reflects a more complex attack surface due to VLAN segmentation.

C. DFS Matrix and Simplification of the Attack Tree

The attack tree generated by the MulVAL tool is converted into a matrix that serves as input for the neural networks used in the DQN and QR-DQN models. As the training data becomes more extensive, the input matrix can become larger and more complex. To solve this problem and improve the success rate of the models, we propose to simplify the input matrix using the Depth-First Search (DFS) algorithm.

1) Matrix simplification via DFS: DFS simplifies the matrix by systematically going through each node in the attack tree and exploring each branch as much as possible before going back. This approach ensures that all possible attack paths are considered while eliminating redundant or superfluous nodes that do not make a meaningful contribution to the final attack path. The resulting simplified matrix reduces the computational burden on the models, allowing them to be processed more efficiently.

2) Assignment of rewards: To help the models prioritize critical vulnerabilities, we assign reward values to each node in the matrix based on its importance. For each node with a vulnerability, we use a score (Vul) to represent the reward value. The start node (node 1) is assigned a reward of -1, while the end node (node 26) in scenario 1 is also assigned -1. Non-critical nodes are assigned a reward value of 0, and all nodes without access to another node are also assigned -1.

a) In scenario 1: The matrix has 26 nodes, resulting in a 26x26 matrix that contains all the necessary transitions between the nodes. By simplifying the matrix with DFS, we reduce unnecessary operations before passing them to the DQN model, which saves processing time and improves overall performance. In the QR-DQN model, each node is also assigned a reward value, but the matrix structure and reward assignment are slightly different. In scenario 2, the matrix contains 17 nodes, with a final size of 17x17. The start node (node 2) is assigned a high reward of 100, while the end node (node 17) is assigned a lower reward of 0.20.

b) In Scenario 2: Both models had the same number of nodes (41), but the way they assigned rewards and processed the nodes was different. The DQN model starts with node 1,

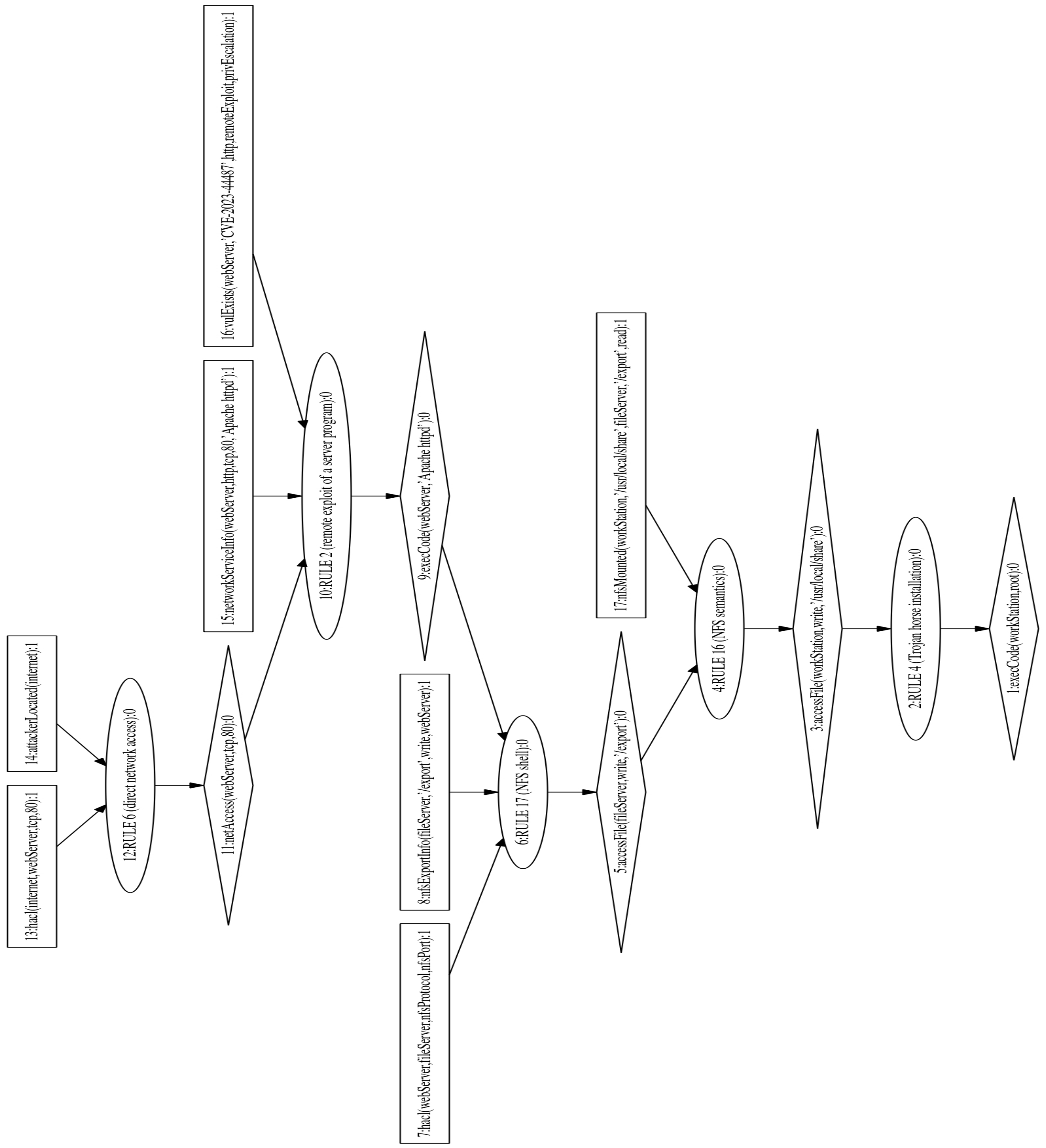

13:hacl(internet,webServer,tcp,80):1
14:attackerLocated(internet):1
12:RULE 6 (direct network access):0
11:netAccess(webServer,tcp,80):0
15:networkServiceInfo(webServer,http,tcp,80,'Apache httpd'):1
16:vulExists(webServer,'CVE-2023-44487',http,remoteExploit,privEscalation):1
10:RULE 2 (remote exploit of a server program):0
9:execCode(webServer,'Apache httpd'):0
8:nfsExportInfo(fileServer,'/export',write,webServer):1
7:hacl(webServer,fileServer,nfsProtocol,nfsPort):1
6:RULE 17 (NFS shell):0
5:accessFile(fileServer,write,'/export'):0
17:nfsMounted(workStation,'/usr/local/share',fileServer,'/export',read):1
4:RULE 16 (NFS semantics):0
3:accessFile(workStation,write,'/usr/local/share'):0
2:RULE 4 (Trojan horse installation):0
1:execCode(workStation,root):0

Fig. 6. Attack graph in scenario 1.

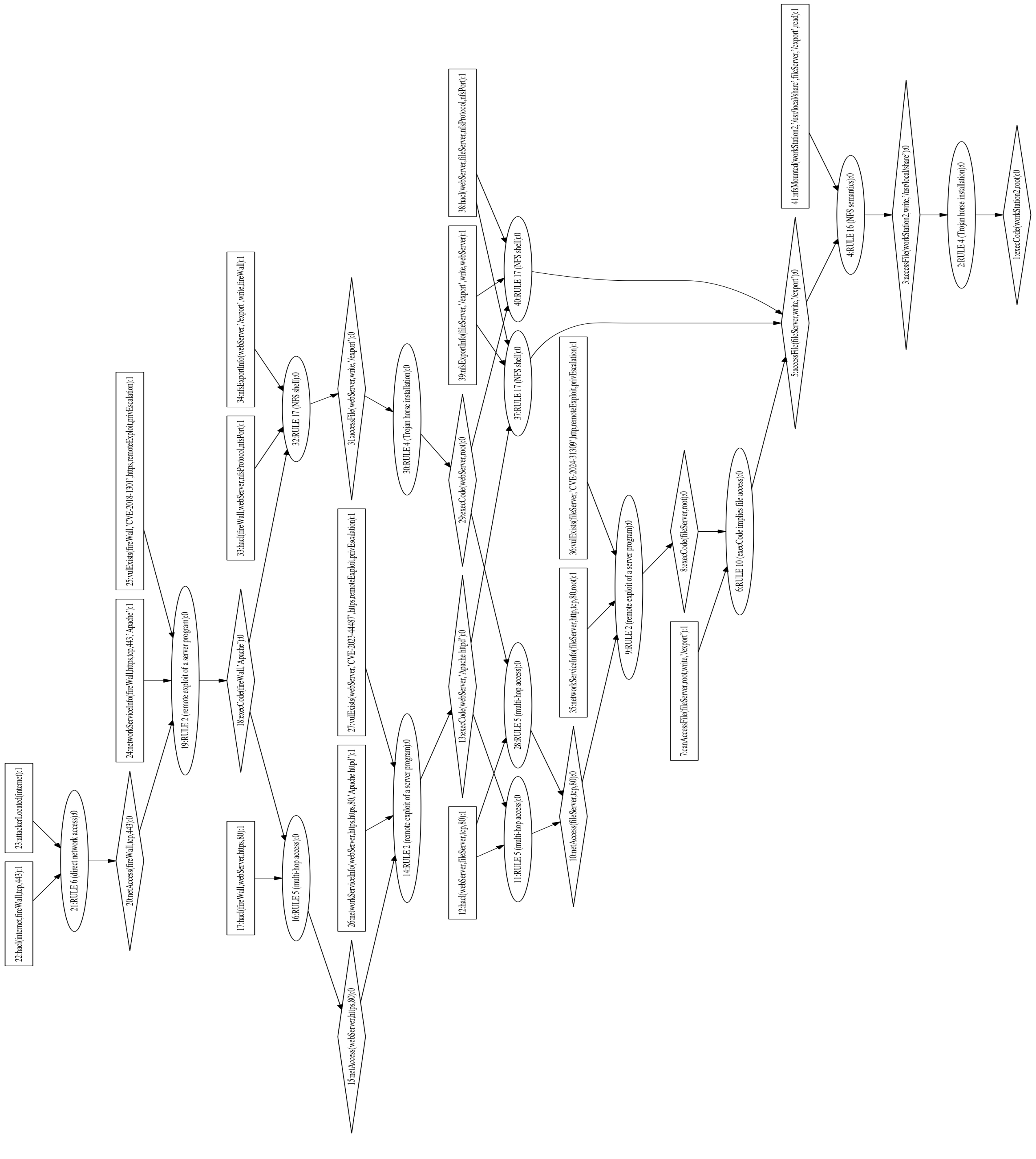

22:hacl(internet,fireWall,tcp,443):1
23:attackerLocated(internet):1
21:RULE 6 (direct network access):0
20:netAccess(fireWall,tcp,443):0
24:networkServiceInfo(fireWall,https,tcp,443,'Apache'):1
25:vulExists(fireWall,'CVE-2018-1301',https,remoteExploit,privEscalation):1
19:RULE 2 (remote exploit of a server program):0
18:execCode(fireWall,'Apache'):0
17:hacl(fireWall,webServer,https,80):1
16:RULE 5 (multi-hop access):0
15:netAccess(webServer,https,80):0
33:hacl(fireWall,webServer,nfsProtocol,nfsPort):1
34:nfsExportInfo(webServer,'/export',write,fireWall):1
32:RULE 17 (NFS shell):0
31:accessFile(webServer,write,'/export'):0
30:RULE 4 (Trojan horse installation):0
29:execCode(webServer,root):0
26:networkServiceInfo(webServer,https,https,80,'Apache httpd'):1
27:vulExists(webServer,'CVE-2023-44487',https,remoteExploit,privEscalation):1
14:RULE 2 (remote exploit of a server program):0
13:execCode(webServer,'Apache httpd'):0
12:hacl(webServer,fileServer,tcp,80):1
11:RULE 5 (multi-hop access):0
28:RULE 5 (multi-hop access):0
10:netAccess(fileServer,tcp,80):0
38:hacl(webServer,fileServer,nfsProtocol,nfsPort):1
39:nfsExportInfo(fileServer,'/export',write,webServer):1
40:RULE 17 (NFS shell):0
37:RULE 17 (NFS shell):0
35:networkServiceInfo(fileServer,http,tcp,80,root):1
36:vulExists(fileServer,'CVE-2024-31309',http,remoteExploit,privEscalation):1
9:RULE 2 (remote exploit of a server program):0
8:execCode(fileServer,root):0
7:canAccessFile(fileServer,root,write,'/export'):1
6:RULE 10 (execCode implies file access):0
5:accessFile(fileServer,write,'/export'):0
41:nfsMounted(workStation2,'/usr/local/share',fileServer,'/export',read):1
4:RULE 16 (NFS semantics):0
3:accessFile(workStation2,write,'/usr/local/share'):0
2:RULE 4 (Trojan horse installation):0
1:execCode(workStation2,root):0

Fig. 7. Attack graph in scenario 2.

assigns it a reward of -1, and ends with node 41, which also has a reward of -1. In contrast, the QR-DQN model starts with node 2 and assigns it a reward of 100 and ends with node 41, which is assigned a reward of 0.20. These differences result from how each model interprets and learns from the state representations. DQN focuses on identifying the most immediate rewards and optimal paths in a direct way, while QR-DQN considers the distribution of rewards, allowing it to explore deeper and more uncertain paths that may offer higher rewards in the long run.

D. DQN /QRDQN Dataset Generation

To train both the DQN and QR-DQN models, datasets were created by defining the network environment, enumerating possible actions, simulating transitions, and assigning rewards. These datasets enabled the models to learn and optimize the attack paths in the penetration test scenarios. The dataset creation includes information about hosts, vulnerabilities, and services so that the models can simulate how attackers could exploit vulnerabilities in the network. Each host in the network is associated with specific vulnerabilities that the models can exploit to extend their reach. The following table summarizes the key vulnerabilities, products, ports, and protocols used in the dataset (Table II):

TABLE II. DQN/QR-DQN DATASET

Host	Vulnerability	Product	Port	Protocol
Web Server	CVE-2023-44487	Apache	80	HTTP
File Server	CVE-2024-31309	Apache	21	FTP
Workstation	——	———	-	HTTP

CVE-2023-44487 (Web Server) is a critical vulnerability that allows Privilege Escalation and Denial of Service (DoS) attacks via remote code execution. CVE-2024-31309 (File Server) is a user-level impact vulnerability that restricts certain actions to lower privileges.

The datasets allow the DQN and QR-DQN models to learn how to exploit vulnerabilities such as CVE-2023-44487 to execute optimal attack paths. For QR-DQN, additional reward distributions were generated to account for uncertainty, allowing the model to explore a wider range of possible outcomes.

E. DQN/QR-DQN Model and Training Results

After creating the input datasets, we trained both the DQN and QR-DQN models. Below is a description of the architecture and training process for each model:

1) DQN/QR-DQN Model:

a) The DQN model: Uses 64 features that provide a good balance between computational efficiency and sufficient complexity to handle attack paths in moderate environments. This feature size ensures that the model can learn quickly and still capture important details about network status and vulnerabilities. The model consists of three layers: two linear layers and a batch normalization layer. The first layer converts the input into 64 features, while the second layer converts it into the final output, which represents the possible attack paths. This architecture is efficient for environments such as scenario 1, where the network is relatively simple.

b) The QR-DQN model: Uses 128 features to handle more complex environments such as Scenario 2, where deeper exploration and uncertainty in the reward distribution must be considered. The larger number of features allows the model to capture more detailed information about possible attack paths and outcomes. The architecture comprises two linear layers and a stack normalization. The first layer converts the input state into 128 features, the second layer retains these features, and the last layer outputs the Q-values for all actions and quantiles. This added complexity helps QR-DQN explore more potential paths, especially in larger, more complex network environments.

2) DQN/QR-DQN Training results:

a) In scenario 1: both models were tested in a simple WLAN setup with three hosts and a web server vulnerability.

The DQN model identified the following optimal attack path, which is:

$$23 \rightarrow 21 \rightarrow 20 \rightarrow 19 \rightarrow 18 \rightarrow 16 \rightarrow 15 \rightarrow 14 \rightarrow 13$$
$$\rightarrow 37 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$$

The QR-DQN model examined a slightly more detailed attack path:

$$23 \rightarrow 21 \rightarrow 20 \rightarrow 19 \rightarrow 18 \rightarrow 32 \rightarrow 31 \rightarrow 30 \rightarrow 29 \rightarrow 28$$
$$\rightarrow 10 \rightarrow 9 \rightarrow 8 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$$

The difference in the paths shows QR-DQN's ability to explore more comprehensive paths that account for uncertainties and additional vulnerabilities.

b) In scenario 2: the complexity of the network increased due to multiple subnets and VLAN segmentation. Both models were able to calculate attack paths but with different levels of detail.

The DQN model identified the following attack path:

$$23 \rightarrow 21 \rightarrow 20 \rightarrow 19 \rightarrow 18 \rightarrow 32 \rightarrow 31 \rightarrow 30 \rightarrow 29$$
$$\rightarrow 28 \rightarrow 10 \rightarrow 9 \rightarrow 8 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$$

The QR-DQN model has calculated a more detailed attack path:

$$23 \rightarrow 21 \rightarrow 20 \rightarrow 19 \rightarrow 18 \rightarrow 32 \rightarrow 31 \rightarrow 30$$
$$\rightarrow 29 \rightarrow 28 \rightarrow 10 \rightarrow 9 \rightarrow 8 \rightarrow 6 \rightarrow 5 \rightarrow 4$$
$$\rightarrow 3 \rightarrow 2 \rightarrow 1$$

In more complex environments, the QR-DQN model explored more attack paths using 128 features, while the DQN model identified a more direct path with 64 features.

X. Performance Analysis of Automated DoS Penetration Testing

In this study, the performance of the DQN and QR-DQN models was evaluated using two main groups of metrics with the same hyperparameter values, as shown in Tables III and IV.

1) Time-Related metrics: duration of the episode, rewards, and mean steps per episode.

2) Performance metrics: Accuracy, precision, recall, F1 score and total time spent.

TABLE III. Hyper-Parameter Values of DQN Algorithm

Hyperparameter	Value
BATCH_SIZE	64
GAMMA	0.98
EPS START	0.99
EPS END	0.01
EPS DECAY	2000
TARGET UPDATE	5
N ACTIONS	Value from file
N STATES	10

TABLE IV. Hyper-Parameter Values of QR-DQN Algorithm

Hyperparameter	Value
BATCH_SIZE	64
GAMMA	0.98
EPS START	0.99
EPS END	0.01
EPS DECAY	2000
TARGET UPDATE	5
N QUANTILES	100

A. Time-Related Metrics

Time-related metrics provide information on how efficiently and quickly the models have explored the attack surface and identified vulnerabilities.

1) Episode duration: Indicates how long each episode lasted. A longer duration indicates a more thorough exploration or a deeper investigation.

2) Rewards: Higher rewards indicate the model's success in finding efficient attack paths. Fluctuations in rewards reflect variability in the discovery of attack paths.

3) Mean steps per episode: Fewer steps indicate more efficient strategies, as the model requires fewer actions to achieve its objectives.

a) In scenario 1: the DQN model converged faster, with increasing episode duration, suggesting that the agent engaged in more challenging tasks and refined its approach, peaking as 3000, while QR-DQN took shorter, with episode duration peaking at 5000 episodes. The DQN model achieved higher rewards 300000 with some variability, reflecting better performance, as shown in Fig. 9, while QR-DQN achieved lower but more consistent rewards 25,000. Although QR-DQN focuses on efficiency and adaptability, it sacrifices some reward maximization compared to DQN. DQN started low, with a sharp increase in the middle episodes, and peaked at over 20,000 steps per episode, while QR-DQN started low, increased to a peak of around 8 steps in the middle episodes, and stabilized at around 2-3 steps towards the end. This reflects the superior efficiency of QR-DQN in identifying optimal paths with minimal exploration, as shown in Fig. 10. Fig. 8 clearly shows the difference between the two models.

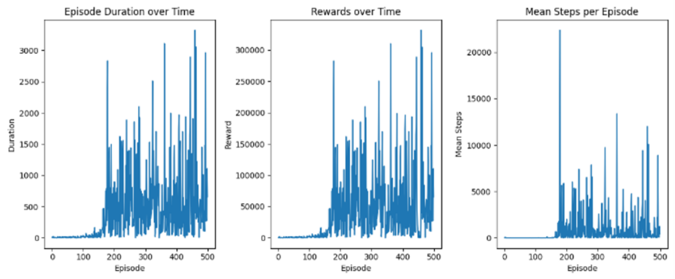


Fig. 9. Experimental results for the DQN network model in scenario 1.

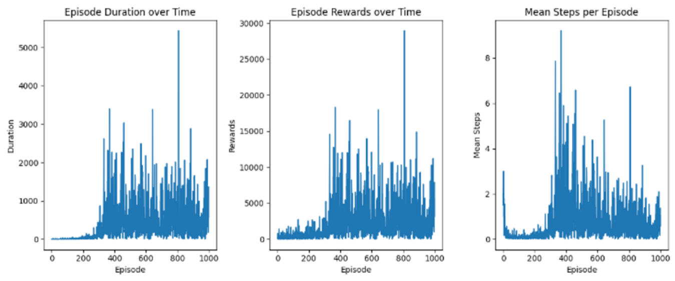


Fig. 10. Experimental results for QR-DQN network model in scenario 1.

b) In scenario 2: with a more complex network environment, both models had a longer episode duration. DQN peaked at 4000 episodes, while QR-DQN peaked at 7000 episodes. DQN achieved higher peak rewards 400000 but with higher variability, while QR-DQN's rewards peaked at 350000, with greater consistency. The average steps per episode for DQN initially peaked at 35000, while QR-DQN steps per episode peaked at 8 steps. Fig. 12 and Fig. 13 illustrate that. Fig. 11 clearly shows the difference between the two models.

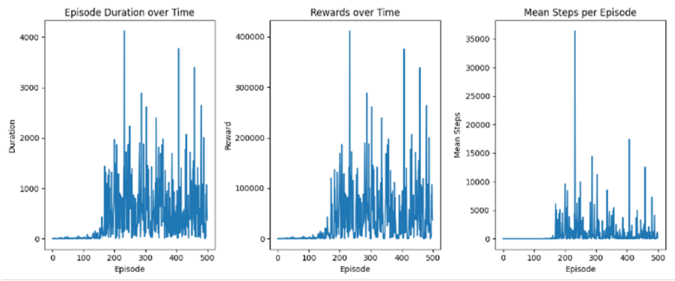
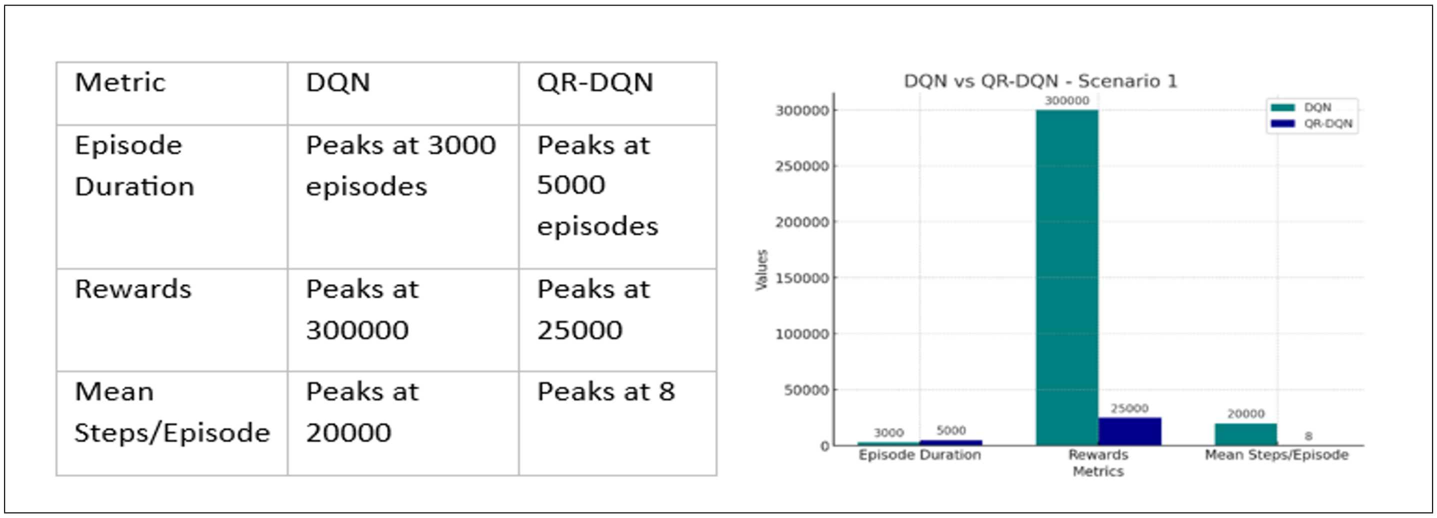
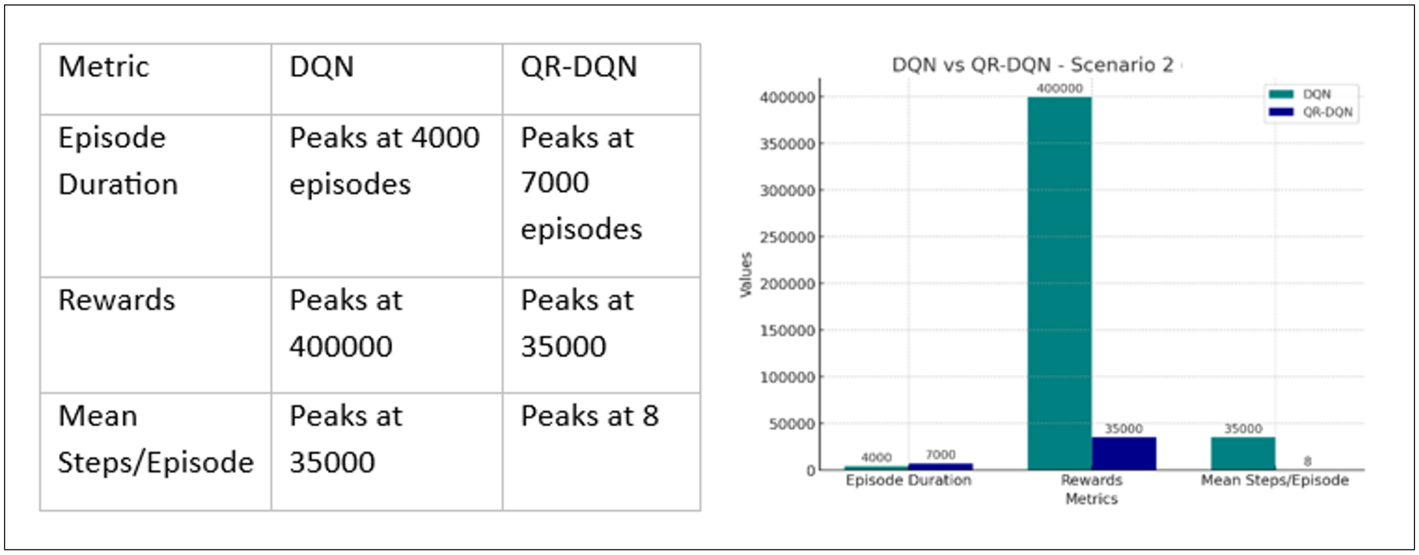


Fig. 12. Experimental results for the DQN network model in scenario 2.



Metric	DQN	QR-DQN
Episode Duration	Peaks at 3000 episodes	Peaks at 5000 episodes
Rewards	Peaks at 300000	Peaks at 25000
Mean Steps/Episode	Peaks at 20000	Peaks at 8

Fig. 8. DQN vs QR-DQN in scenario 1.



Metric	DQN	QR-DQN
Episode Duration	Peaks at 4000 episodes	Peaks at 7000 episodes
Rewards	Peaks at 400000	Peaks at 35000
Mean Steps/Episode	Peaks at 35000	Peaks at 8

Fig. 11. DQN vs QR-DQN in scenario 2.

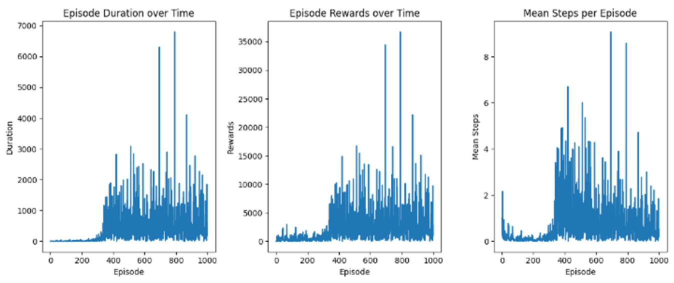


Fig. 13. Experimental results for the QR-DQN network model in scenario 2.

B. Performance Metrics

These metrics evaluate the ability of the models to correctly predict attack paths and accurately exploit vulnerabilities, taking into account efficiency over time.

1) Accuracy: The proportion of correct predictions (attack paths) made by the model.

2) Precision: The ability of the model to correctly identify the correct vulnerabilities out of all predicted vulnerabilities.

3) F1 score: A balanced measure that combines both precision and recall and is useful for evaluating overall performance.

4) Total time: The total time taken by the model to train and identify the attack paths.

a) In scenario 1: The DQN model achieved very powerful metrics with an accuracy of 99%, a precision of 100%, a recall of 99%, and an F1 score of 100%, indicating excellent precision and detection in the simpler network configuration. The total time for training and attack detection was 2.01 seconds, reflecting the faster convergence of the DQN in simpler environments. The QR-DQN model also achieved a high accuracy of 99% but a precision of 100%, a recall of 99%, and an F1 score of 99%. Both models achieve high accuracy, precision, and recall, demonstrating their ability to effectively detect attack paths. The following table clearly shows the difference between the two models in the performance matrices as shown in Table V.

TABLE V. Scenario 1 DQN vs QR-DQN Performance (Accuracy, Precision, F1-Score, Time)

Metric	DQN	QR-DQN
Accuracy (%)	99	99
Precision (%)	100	100
Recall (%)	99	99
F1-Score (%)	100	99
Total Time	2.01 seconds	4.38 seconds

5) In scenario 2: Both models have maintained their strong performance. DQN achieved an accuracy of 98 %, a precision of 100%, a recognition of 98 %, and an F1 score of 99 %, completing training in 1.61 seconds. QR-DQN achieved the same precision 100%, accuracy 99 %, recall 99%, and F1 score of 99%, and a training time of 1.98 seconds due to the deeper exploration of the complex network environment. Table VI clearly shows the difference between the two models in the performance matrices.

TABLE VI. Scenario 2 DQN vs QR-DQN Performance (Accuracy, Precision, F1-Score, Time)

Metric	DQN	QR-DQN
Accuracy (%)	98	99
Precision (%)	100	100
Recall (%)	98	99
F1-Score (%)	99	99
Total Time	1.61 seconds	1.98 seconds

XI. Discussion

Several important findings emerge from the evaluation. They are as follows:

1) DQN shows faster convergence: in both scenarios with higher rewards and shorter training times, especially in scenario 1, where the network environment is simpler. It is characterized by high accuracy (99%) and precision (100%), which makes it very effective for scenarios that require fast and direct identification of attack paths. While QR-DQN is slower to converge and requires more time to train, it is excellent for complex environments such as Scenario 2, where deeper reconnaissance is required. QR-DQN's ability to model reward uncertainty results in more consistent rewards and higher F1 scores (99%) in both scenarios, ensuring fewer false alarms and balanced performance between accuracy and thoroughness.

2) Episode duration and steps: The QR-DQN model consistently exhibited longer episode duration and required more steps initially, reflecting its thorough exploration process. However, it eventually stabilized at the same level of efficiency as the DQN model, making it more suitable for more complex penetration testing scenarios where exploration of insecure attack paths is critical.

3) Trade-off between time and accuracy: DQN is faster and achieves high accuracy and efficiency in simpler scenarios, but QR-DQN offers more stability and reliability when dealing with uncertainty, but at the cost of a longer training time.

The above statistics show that the QR-DQN model is preferable for automated penetration testing as it has consistent performance in terms of longer episodes and a more consistent accumulation of rewards, suggesting that it is more comprehensive and trustworthy when investigating and exploiting vulnerabilities. The constant stabilization of mean steps per episode demonstrates the efficiency of QR-DQN throughout the testing process. Scenario 2 showed higher episode duration, higher rewards, and greater variation in average steps per episode in both algorithms, suggesting that the agent in scenario 2 explores more, achieves higher rewards, and encounters more variability on its path to optimal solutions. This suggests that the design or parameters of Scenario 2 encourage deeper exploration and learning compared to Scenario 1.

The observed faster reward stability and shorter episode duration in QR-DQN have direct consequences for practice. Faster incentive accumulation leads to faster detection of major vulnerabilities, allowing organizations to reduce risks more effectively. Shorter episode times enable faster decision-making and less system downtime during penetration testing, resulting in less disruption without compromising network security.

XII. Conclusion

This study investigated the feasibility of using a Deep Q-learning Network (DQN) and a Quantile Regression Deep Q-network (QR-DQN) for automated attack path planning in penetration testing, using MulVAL for sparse rewards. The results show that the DQN learns faster, with peak rewards of 300,000 in scenario 1 and 400,000 in scenario 2. However, its aggressive exploration led to high variance, resulting in unstable learning behavior and lower steady-state rewards.

In contrast, QR-DQN showed more stable performance by effectively modeling reward uncertainty. Although the peak rewards of QR-DQN were lower (250,000 in Scenario 1 and 350,000 in Scenario 2), it provided more reliable exploration in complex scenarios. However, this stability came at the cost of a longer learning time — QR-DQN required approximately 5,000 episodes to reach its performance peak in Scenario 1, compared to 3,000 episodes for DQN and 7,000 episodes in Scenario 2, compared to 4,000 episodes for DQN. In addition, QR-DQN performed significantly fewer steps per episode, at least eight, compared to DQN's 20,000 in Scenario 1, indicating more efficient path planning. Despite QR-DQN's advantages in terms of stability and structured exploration, its time-intensive nature remains a limitation. Future improvements will focus on optimizing QR-DQN to balance efficient exploration and reduced computational effort.

XIII. Future Works

Future research will focus on integrating real-time data to improve the system's adaptability in responding to dynamic threats. The model will be trained with historical vulnerability data using machine learning techniques that enable improved predictive capabilities and proactive threat defense.

In addition, implementing broader simulations, especially in IoT environments and large infrastructures, will be explored to assess the model's scalability and improve its generalization to different network architectures. Extending the experimental framework to include comparisons with advanced DRL algorithms, such as Advantage Actor-Critic (A3C) and Proximal Policy Optimization (PPO), will provide deeper insights into the relative strengths and limitations of QR-DQN. These comparisons will help refine the model's efficiency and evaluate its effectiveness in penetration testing scenarios, contributing to the development of more robust and resilient automated security assessment systems.

ACKNOWLEDGMENT

This work was supported by the Deanship of Scientific Research, Vice Presidency for Graduate Studies and Scientific Research, King Faisal University, Saudi Arabia [Project No. KFU250998].

AUTHORS' CONTRIBUTIONS

Both authors equally contributed.

REFERENCES

[1] Hu Z, Beuran R, Tan Y. Automated penetration testing using deep reinforcement learning. In: 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). IEEE; 2020. p. 2-10.

[2] Maeda R, Mimura M. Automating post-exploitation with deep reinforcement learning. Comput Secur. 2021;100:102108.

[3] Al-Saraireh JM. Enhancing the penetration testing approach and detecting advanced persistent threat using machine learning [PhD thesis]. Princess Sumaya University for Technology; 2021.

[4] Goh KC. Toward automated penetration testing intelligently with reinforcement learning [PhD thesis]. Dublin: National College of Ireland; 2021.

[5] Huizinga T. Using machine learning in network traffic analysis for penetration testing auditability. 2019.

[6] Chu G, Lisitsa A. Poster: Agent-based (BDI) modeling for automation of penetration testing. In: 2018 16th Annual Conference on Privacy, Security and Trust (PST). IEEE; 2018. p. 1-2.

[7] Sommervoll ÅÅ, Erdődi L, Zennaro FM. Simulating all archetypes of SQL injection vulnerability exploitation using reinforcement learning agents. Int J Inf Secur. 2024;23(1):225-246.

[8] Tran K, Akella A, Standen M, Kim J, Bowman D, Richer T, et al. Deep hierarchical reinforcement agents for automated penetration testing. arXiv preprint arXiv:2109.06449. 2021.

[9] Koroniotis N, Moustafa N, Turnbull B, Schiliro F, Gauravaram P, Janicke H. A deep learning-based penetration testing framework for vulnerability identification in Internet of Things environments. In: 2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). IEEE; 2021. p. 887-894.

[10] Kujanpää K, Victor W, Ilin A. Automating privilege escalation with deep reinforcement learning. In: Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security; 2021. p. 157-168.

[11] Neal C, Dagdougui H, Lodi A, Fernandez JM. Reinforcement learning based penetration testing of a microgrid control algorithm. In: 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC). IEEE; 2021. p. 0038-0044.

[12] Semenov S, Weilin C, Liqiang Z, Bulba S. Automated penetration testing method using deep machine learning technology. 2021.

[13] Tran K, Standen M, Kim J, Bowman D, Richer T, Akella A, et al. Cascaded reinforcement learning agents for large action spaces in autonomous penetration testing. Appl Sci. 2022;12(21):11265.

[14] Zennaro FM, Erdodi L. Modelling penetration testing with reinforcement learning using capture-the-flag challenges: Trade-offs between model-free learning and a priori knowledge. IET Inf Secur. 2023.

[15] Zhou S, Liu J, Hou D, Zhong X, Zhang Y. Autonomous penetration testing based on improved deep Q-network. Appl Sci. 2021;11(19):8823.

[16] Gangupantulu R, Cody T, Park P, Rahman A, Eisenbeiser L, Radke D, et al. Using cyber terrain in reinforcement learning for penetration testing. In: 2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS). IEEE; 2022. p. 1-8.

[17] Chowdhary A, Huang D, Mahendran JS, Romo D, Deng Y, Sabur A. Autonomous security analysis and penetration testing. In: 2020 16th International Conference on Mobility, Sensing and Networking (MSN). IEEE; 2020. p. 508-515.

[18] Zhang Y, Liu J, Zhou S, Hou D, Zhong X, Lu C. Improved deep recurrent Q-network of POMDPs for automated penetration testing. Appl Sci. 2022;12(20):10339.

[19] Yang Y, Liu X. Behaviour-diverse automatic penetration testing: A curiosity-driven multi-objective deep reinforcement learning approach. arXiv preprint arXiv:2202.10630. 2022.

[20] Sangamesvarappa V. Parallelizing Depth-First Search for Pathway Finding: A Comprehensive Investigation. Revue d'Intelligence Artificielle. 2023;37(4):123-145.

[21] Chen Z, Kang F, Xiong X, Shu H. A Survey on Penetration Path Planning in Automated Penetration Testing. Applied Sciences. 2024; 14(18):8355.