applied sciences

MDPI

Article

Automated Vulnerability Exploitation Using Deep Reinforcement Learning

Anas AlMajali [1,2,*], Loiy Al-Abed [1], Khalil M. Ahmad Yousef [1], Bassam J. Mohd [1], Zaid Samamah [1], and Anas Abu Shhadeh [1]

[1] Department of Computer Engineering, The Hashemite University, Zarqa 13115, Jordan; luaiehsan@outlook.com (L.A.-A.); khalil@hu.edu.jo (K.M.A.Y.); bassam@hu.edu.jo (B.J.M.); zsamamah@yahoo.com (Z.S.); anas.essa79@gmail.com (A.A.S.)

[2] Department of Computer Science and Engineering, American University of Sharjah, Sharjah 26666, United Arab Emirates

[*] Correspondence: almajali@hu.edu.jo or aalmajali@aus.edu

Abstract: The main objective of this paper is to develop a reinforcement agent capable of effectively exploiting a specific vulnerability. Automating pentesting can reduce the cost and time of the operation. While there are existing tools like Metasploit Pro that offer automated exploitation capabilities, they often require significant execution times and resources due to their reliance on exhaustive payload testing. In this paper, we have created a deep reinforcement agent specifically configured to exploit a targeted vulnerability. Through a training phase, the agent learns and stores payloads along with their corresponding reward values in a neural network. When encountering a specific combination of a target operating system and vulnerability, the agent utilizes its neural network to determine the optimal exploitation options. The novelty of this work lies in employing Deep Reinforcement Learning in vulnerability exploitation analysis. To evaluate our proposed methodology, we conducted training and testing on the Metasploitable platform. The training phase of the reinforcement agent was conducted on two use cases: the first one has one vulnerability, and the second one has four vulnerabilities. Our approach successfully achieved the attacker's primary objective of establishing a reverse shell with a maximum accuracy of 96.6% and 73.6% for use cases one and two, respectively.

Keywords: cybersecurity; vulnerability exploitation; penetration testing; risk assessment; reinforcement learning; neural networks; machine learning

check for updates

Citation: AlMajali, A.; Al-Abed, L.; Ahmad Yousef, K.M.; Mohd, B.J.; Samamah, Z.; Abu Shhadeh, A. Automated Vulnerability Exploitation Using Deep Reinforcement Learning. *Appl. Sci.* **2024**, *14*, 9331. https://doi.org/10.3390/app14209331

Academic Editors: Xavier A. Larriva-Novo and Andres Marin Lopez

Received: 28 July 2024
Revised: 10 October 2024
Accepted: 10 October 2024
Published: 13 October 2024

1. Introduction

Everyday, Information Technologies (IT) and Operation Technologies (OT) evolve to provide more services to humans. On the other hand, new vulnerabilities and threats appear in our systems, making them susceptible to cyber attacks. The arms race between attackers and defenders will never stop. The great advancements in Artificial Intelligence (AI) and Machine Learning (ML) helped solve problems that surpass human capabilities like winning a game of GO [1].

Cybersecurity risk assessment can be conducted by experts to assess the security posture of an organization. This process involves vulnerability and threat analysis, and it is part of a larger process, which is risk management. The evaluation of vulnerabilities in a system can be performed manually by experts or automatically using special tools and algorithms. ML techniques can be utilized to automate vulnerability assessment and evaluation, which reduces the cost and effort (time) required by security experts. The impact of exploiting vulnerabilities can be catastrophic, affecting the confidentiality, integrity, and availability of IT and OT systems. The impact can include financial loss, reputational damage, and even physical harm [2].

By identifying vulnerabilities in an organization's systems and networks, organizations can implement patches, upgrade software, or modify configuration settings to improve the security posture of their systems. On average, out of the total number of vulnerabilities reported by vulnerability scanners, only 82% were relevant results (identified correctly), regardless of the vulnerabilities that scanners failed to report (18% were false positives) [3]. Penetration testing provides organizations with a comprehensive evaluation of their security posture by identifying potential weaknesses or vulnerabilities in their systems and networks. This allows organizations to prioritize their efforts to improve security and mitigate risk.

Reinforcement Learning (RL) algorithms provide a framework for the agent to learn from its own experiences, by trial and error, to determine the optimal policy to follow in a given situation. The approach taken by RL algorithms is inherently different from supervised and unsupervised learning, as it involves a sequential decision-making process, where the outcome of each action is dependent on the previous decisions made. Traditional model-based approaches may fail in complex and changing environments. This makes RL especially important, as it is capable of dealing with such environments. RL algorithms can continuously learn from new situations and enhance their decision accordingly. This makes RL irreplaceable for dealing with situations where the optimal solution is uncertain.

In this paper, our main objective is to utilize the power of AI in automating the process of vulnerability exploitation. This can be accomplished by creating an AI agent that performs penetration testing scenarios using a large number of payloads that are available in the Metasploit framework [4]. By doing so, we aim to successfully identify the most efficient payloads that can be used to exploit certain vulnerabilities. The identified most efficient payloads can then be used to test vulnerabilities in other systems. Automating this process using the power of AI and ML can significantly improve the efficiency of the risk assessment process, which leads to improving risk management. Therefore, the novelty of this work is employing Deep Reinforcement Learning (DRL) in vulnerability exploitation analysis.

The RL agents are trained to conduct penetration testing by attempting to exploit vulnerabilities using their payloads from the Metasploit framework. The agent is encouraged to pursue successful exploitation through a reward system, where successfully establishing a session on the target machine through a given payload results in a high reward, and failure to do so results in a low reward. The ultimate goal is to determine the most effective payload for exploiting identified vulnerabilities and analyzing their exploitability.

The main contributions of this paper are as follows:

1. Implemented vulnerability scanning supported by RL vulnerability exploitation analysis (Section 4).
2. Designed, trained, and implemented an RL agent to perform vulnerability exploitation analysis by leveraging DRL, which performed better than Q-Tables in terms of accuracy. DRL achieved a maximum accuracy of 96.6% for a single vulnerability exploitation.
3. Tested the DRL learned model on multiple vulnerability scenarios.
4. Compared the results of using Q-Table versus using DRL in vulnerability exploitation analysis.

The rest of the paper is organized as follows. Section 2 lays out the necessary background needed to understand our proposed DRL agent. Section 3 summarizes the latest research work in automated penetration testing and the combination of AI, ML, and security risk assessment. Section 4 presents the detailed research methodology. Section 5 summarizes and discusses the results. Section 6 discusses concluding remarks and future research opportunities.

2. Background

In this section, we lay out the background of this work in two subsections that discuss RL and vulnerability exploitation.

2.1. *Reinforcement Learning Fundamentals*

Machine Learning (ML) has many subfields, including RL, which deals with the problem of an agent that learns by trial and error (rewarded actions) to make decisions in an environment [5]. The RL agent takes certain actions to get closer to a certain objective. The agent receives feedback in the form of a reward for each action taken, favoring actions with a high reward and updating the decision-making policy accordingly. Optimally, the agent converges to a policy of states and actions that maximizes the cumulative average reward.

2.1.1. RL Using Q-Table

An RL agent is able to perceive and interpret its environment, takes actions, and learns through trial and error. It does this by calculating Q-values and, later on, storing them in a Q-Table. So, for a given state in an environment, a bunch of Q-values are assigned for each action the agent takes. The Q-values are calculated according to Equation (1).

$$Q(S, A) = (1 - \alpha)Q(S, A) + \alpha(R + \gamma max(Q(S', A'))) \tag{1}$$

where:

- $Q(S, A)$ is the Q-value for state S and action A.
- α is the learning rate, which determines the weight of the new information compared with the existing Q-value. It is typically a value between 0 and 1.
- R is the immediate reward observed after taking action A in state S.
- γ is the discount factor, which determines the importance of future rewards compared with immediate rewards. It is also typically a value between 0 and 1.
- S' is the next state after taking action A in state S.
- A' is the action with the maximum Q-value in the next state S'.
- $max(Q(S', A'))$ represents the maximum Q-value for all possible actions in the next state S'.

2.1.2. RL Using Deep Neural Networks

While the Q-Table is a very efficient method to store the estimated Q-value for each state-action pair, it faces performance problems when dealing with large or continuous state spaces. Storing large states' data overwhelms the memory, and exploring every single combination becomes impractical. Such problems are addressed by Deep Q-Network (DQN), where Q-Table is replaced by NN. The NN takes the state as input and outputs the Q-values for all possible actions. The DQN is capable of generalizing from the learned patterns to explore unseen states and it has many useful applications like speech recognition, natural language processing, and malware detection [6,7]. However, DQN designers should be aware that training can be computationally expensive and susceptible to overfitting.

The standard DQN employs a single NN, which replaces the Q-Table. However, single-NN DQN can experience stability issues during training. To address this, DQN actually leverages two NNs: a policy network and a target network, as shown in Figure 1. The policy network is responsible for making decisions based on the current state of the environment. It takes the state as input and outputs a Q-value for each possible action. The agent then chooses the action with the highest Q-value. The target network is a more stable version of the policy network. It is periodically updated with the weights of the policy network, but not at every training step. This helps to smooth out the learning process.

In the DQN model, the agent interacts with the environment through actions and receiving rewards. Each interaction creates an experience: e_t = (s_t, a_t, r_{t+1}, s_{t+1})

where

- e_t: experience/interaction at time step t;
- s_t: state at time step t;
- a_t: action at time step t;
- r_{t+1}: reward at time step $t + 1$;

- s_{t+1}: next-state (i.e. state at time step $t + 1$).

Also, a DQN model uses a replay memory, which stores the agent's experiences. The following steps are executed during training:

- A small random sample of experiences is sampled from the replay memory.
- For each experience,
  - The target network estimates the optimal Q-value for the next-state from the sampled experiences. This is performed by applying s_{t+1} as input to the target network and then selecting the maximum $q(s', a')$. The result is the target Q-value (i.e., optimal Q-value).
  - The policy network computes the estimated (i.e., predicted) Q-value (i.e., $q(s, a)$). This is achieved by applying s_t as input to the network and then selecting the Q-value corresponding to a_t.
  - The difference (loss) between the predicted Q-value from the policy network and the target Q-value is calculated.
  - The weights of the policy network are adjusted to minimize the loss, effectively making the policy network's Q-value predictions closer to the more stable target Q-values.
  - Occasionally, the weights of the target network are updated to match those of the policy network. This ensures the target network does not become too outdated.

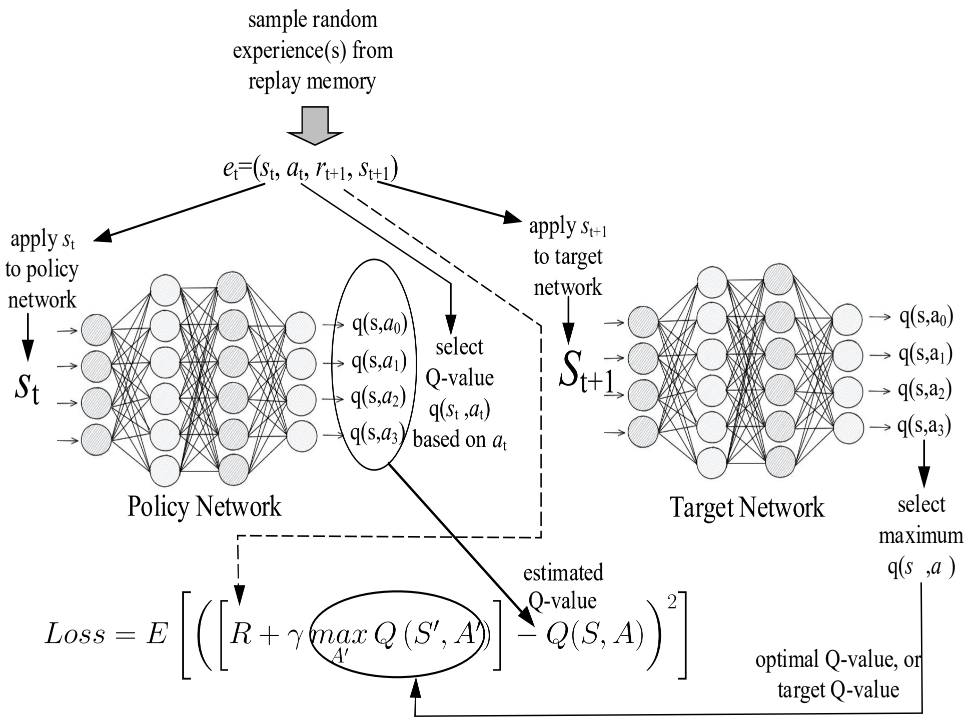

sample random experience(s) from replay memory
$e_t=(s_t, a_t, r_{t+1}, s_{t+1})$
apply s_t to policy network
s_t
q(s,a_0)
q(s,a_1)
q(s,a_2)
q(s,a_3)
select Q-value q(s_t ,a_t) based on a_t
Policy Network
apply s_{t+1} to target network
S_{t+1}
q(s,a0)
q(s,a1)
q(s,a2)
q(s,a3)
Target Network
select maximum q(s ,a)
estimated Q-value
optimal Q-value, or target Q-value
$Loss = E\left[\left(\left[R + \gamma \max_{A'} Q(S', A')\right] - Q(S, A)\right)^2\right]$

Figure 1. Deep Q-Networks (DQN).

It is important to mention that in this work, we opted for a single NN for policy learning. Employing two NNs offered minimal performance improvements compared with the increased complexity, and we did not experience any instability issues during training.

Additionally, to maintain simplicity in our DQN implementation and as Figure 1 shows, we opted for the well-established loss function, denoted as *Loss*, presented in Equation (2):

$$Loss = E\left[\left(\left[R + \gamma \max_{A'} Q(S', A')\right] - Q(S, A)\right)^2\right] \tag{2}$$

where

- $Q(S', A')$ represents the estimated future reward for taking action A' in the next state S', as predicted by the DQN network.
- E denotes the arithmetic mean.

This loss function combines the immediate reward (R) with the discounted estimate of the future reward ($\gamma max(Q(S', A'))$), guiding the agent to learn actions that maximize long-term rewards, leading to successful exploitation.

2.2. Vulnerability Exploitation

Penetration testing is a test methodology in which assessors, typically working under specific constraints, attempt to circumvent or defeat the security features of a system [8]. According to [9], penetration testing passes through the following phases: pre-engagement interactions, intelligence gathering, threat modeling, vulnerability analysis, exploitation, post-exploitation, and reporting. Once a vulnerability is identified in a system, it is analyzed and tested for exploitability. Usually, this process is performed by blindly performing a brute force attack that exhausts all possible attack vectors, which is not productive and noisy [9]. Unseen protective measures and changes to the system may cause certain exploits to fail. In this paper, we propose to perform the exploitation step in an automatic and intelligent way using RL techniques.

Vulnerability exploitation is used as input for the next steps in penetration testing. So, its results have to be accurate and efficient. In addition, if a vulnerability is exploitable, then its *severity* can be estimated within the context of the tested system [10].

3. Related Work

The National Institute of Standards and Technology (NIST) assigns high vulnerability severity values to vulnerabilities that are exposed and exploitable [10], which is important to perform accurate risk assessment and management. In 2016, Defense Advanced Research Projects Agency (DARPA) hosted the Cyber Grand Challenge (CGC) Final Event, the world's first all-machine cyber hacking tournament [11]. This event demonstrates the importance of combining vulnerability exploitation and AI techniques to perform penetration testing.

Most of the work that we investigated in the literature which combines machine learning, vulnerability assessment, and penetration testing focused on post-exploitation and attack-path optimization [12–14]. Next, we discuss the most recent related work.

Chaudhary et al. [13] investigated using ML to automate penetration testing, which helps discover vulnerabilities in computer systems. The authors created a training environment where an agent could explore testing a network environment and find sensitive data. By training the agent in various environments, they aimed to make this method adaptable and work in different situations. The authors also suggested that future work could involve training the agent for more advanced tasks, like performing a deep analysis of the system and exploiting additional vulnerabilities.

Maeda and Mimura [15] proposed a new approach that combined deep RL with PowerShell Empire, a tool that attackers use after penetrating the system (post-exploitation). This created intelligent agents that can make decisions according to the compromised system's state. To train these agents, the authors experimented with the following models: A2C, Q-learning, and SARSA. Interestingly, the A2C model was able to gain the highest reward overall. Finally, the authors tested the trained agents in a completely new network environment. The A2C model was particularly successful in gaining administrative control over a critical system component, which is the domain controller.

Instead of performing regular penetration testing , Hu et al. [16] used deep RL to automate the operation. The authors' system works in two stages. First, they use the Shodan search engine to find relevant server data and build a realistic network topology. Then, a tool called MulVAL generates a map of possible attack routes within this network. Traditional search methods are used to analyze this map and identify all potential attack paths. This information is then converted into a format suitable for deep RL algorithms.

In the next stage, a specific deep RL method called DQN takes over. Its goal is to identify an attack path in the network by exploiting certain vulnerabilities. The authors tested their system with thousands of different network scenarios. The DQN method achieved high accuracy in finding the optimal attack path, reaching up to 86%. In the remaining cases, it still provided valid alternative solutions. In addition, the framework could potentially be used in defense training to automatically recreate attacks in a training environment.

Schwartz and Kurniawat [17] explored using a type of AI called model-free RL to automate pentesting. Their approach involved creating a fast and efficient simulation environment to train and test autonomous pentesting agents.

Within this simulator, they tested Q-learning in two forms: a basic table-based version and one that utilizes artificial neural networks. The results were promising: both versions successfully identified the most effective attack paths across various network layouts and sizes, without needing a pre-built model of action behavior. However, these algorithms were only truly effective for smaller networks with a limited number of possible actions. The researchers acknowledged this limitation and called for further development of scalable RL algorithms that can handle larger, more complex networks in more realistic settings.

Ghanem and Chen [18] presented a novel approach to penetration testing that leverages the power of RL. Their idea was to train an AI agent to actively seek out and exploit vulnerabilities in computer systems. To achieve this, they proposed modeling the penetration testing process as a Partially Observed Markov Decision Process (POMDP). This model captures the uncertainty involved in real-world hacking scenarios. The agent would then learn through trial and error, using an external solver to make the best decisions based on the information it gathers. The main benefit of this approach is the potential for automated and regular testing, freeing up human security specialists for other tasks. Additionally, the ability of the AI to learn and adapt could lead to more accurate and reliable penetration testing compared with traditional methods.

While promising, it’s important to note that their research focused on the planning stage of penetration testing, not the entire process. Further development is needed to create a truly comprehensive AI-powered penetration testing system.

Ghanem and Chen [19] proposed an Intelligent Automated Penetration Testing System (IAPTS) to automate penetration testing for small- and medium-size networks using RL. The main objective of this work is to minimize human intervention in the penetration testing process. The system works by integrating with industrial penetration testing modules and learning from human expertise while performing their tasks. The system relies on RL to learn from human expertise, then uses this knowledge to penetrate similar future scenarios. This reduces human errors that result form tiredness, omission, and stress. However, this system requires human expert supervision in the early learning stages. In addition, this approach is not efficient for large networks. In [20], the authors solved the scalability problem by dividing the network being tested hierarchically as a group of clusters and solving each cluster separately.

Zennaro and Erdődi [21] used Capture The Flag (CTF) competitions to analyze the trade-off between model-free learning and a priori knowledge. The authors demonstrate that providing a priori knowledge to the model-free RL agent reduces the complexity of solving the CTF challenges, allowing the challenges to be solved in a reasonable amount of time.

Erdődi et al. [22] proposed a formalization to simulating SQL injection attacks using Q-learning RL agents utilizing two RL algorithms: the standard tabular Q-learning and DQN. The authors model the attack process as a capture-the-flag challenge, formulating it as a Markov decision process and as a reinforcement learning problem. Their agents learn to exploit SQL injection vulnerabilities, not just for a specific scenario, but to develop generalizable policies applicable to performing SQL injection attacks against any system. The authors analyze the effectiveness and convergence speed of the learned policies against challenges with varying complexity and the learning agent’s complexity. The simulation results provide a proof-of-concept support for using RL agents to perform autonomous penetration testing and security assessment.

Tran et al. [23] proposed an architecture called Cascaded Reinforcement Learning Agents (CRLA) to address the challenge of large action spaces encountered in autonomous penetration testing. The authors formulated their problem as a discrete-time RL task modeled by a Markov decision process (MDP). The proposed RL architecture leverages an algebraic action decomposition strategy, which involves hierarchically structuring RL agents, each tasked with learning within a smaller action subset while still receiving the same external reward signal. This model-free approach eliminates the need for domain knowledge in action decomposition, enabling CRLA to efficiently navigate large action spaces and find optimal attack policies faster and more stably than single DQN agents. The authors use simulated environments from CybORG in a variety of scenarios with different configurations of hosts and action spaces to test their architecture, where all showed that CRLA had superior performance compared with the baseline single-agent Dueling DQN (DDQN), which is the core RL algorithm the authors used in their work.

Yi and Liu [24] proposed an algorithm called MDDQN, which integrates one of the attack graph tools, the multi-stage vulnerability analysis language (MulVAL) and DDQN algorithm for intelligent penetration testing path design, to address the limitations of previous methods.

The authors' experimental results show that the MDDQN algorithm improves the convergence speed and attack path planning efficiency. However, the MDDQN algorithm cannot autonomously scan the constructs and access network information. This means that MDDQN relies on an external source to provide this information, which can limit its effectiveness in real-world scenarios.

Unlike previous research that investigated and analyzed post-exploitation stages of the "Cyber Kill Chain", our approach uses RL to focus on the earlier, pre-exploitation stage. Particularly, we trained an AI agent to pick the right payload to exploit the system. Given an operating system (OS) and a vulnerability, the agent can choose the most effective payload from the Metasploit framework to establish a remote connection between the victim and the attacker. This process has the potential to make security assessments faster and more accurate. By automating payload selection, our method goes beyond simply identifying a vulnerability; it checks if it can actually be exploited. Overall, our research contributes to the field of vulnerability exploitation and security assessments by providing a unique solution to the vulnerability exploitation stage of the security assessment cycle.

4. Methodology

In this section, we combine the details of our experimental setup with a breakdown of the main methods we employed.

4.1. Experimental Setup

In this subsection, we discuss the experimental setup that we used to test two use cases: CouchDB and a Group of Vulnerabilities (GV). Table 1 shows the specification of the machines that were used in the testing and training of the use cases presented in the following subsections.

Table 1. Testing and training simulation setup.

Attacker	Victim
2 GB of RAM DDR4	4 GB of RAM DDR4
1 CPU, 2 Cores (AMD Ryzen 7 4th gen) 2.90 GHz	1 CPU, 2 Cores (AMD Ryzen 7 4th gen) 2.90 GHz
Kali Linux 6.1.0 64-bit	Windows 10 64-bit

4.1.1. Use Case 1, CouchDB

In our previous work [25], RL and Q-Table were used to perform vulnerability exploitation. We specifically tested our approach on Apache CouchDB version 3.1.0 [26], which is vulnerable to remote code execution attacks [27]. This vulnerability is a major security concern because it could be widely exploited by attackers.

To train our RL agent, we used a virtual machine to emulate the attacker that has the Kali Linux OS and another virtual machine which emulates the victim that has the Windows 10 OS. This setup simulates a real-world attack scenario. We then used a different setup for the deployment phase, in which the victim machine has the Windows 11 OS. In this work and for this use case, we used the same setup to test the same vulnerability (i.e., CouchDB), but this time using DRL instead of Q-Table. This single vulnerability with 194 payloads that are designed to exploit it serves as a good use case to test the two RL techniques (Q-Table and DRL).

4.1.2. Use Case 2, Group of Vulnerabilities

In this use case, a Group of Vulnerabilities (GV) was used to test the RL agent. This GV is listed in Table 2. What is common between those vulnerabilities is that they all allow remote code execution by the attacker. A total of 256 payloads were used to train the model. Those payloads were used to try and exploit each one of the vulnerabilities listed in Table 2 in an effort to create an RL agent that can be generalized to a group of vulnerabilities. Q-Table and DRL were separately used to train the model and deploy it later. Having a GV is more challenging to the model, as the state now has more vulnerabilities and more actions to choose from. The actions are the payloads that are used for vulnerability exploitation.

In use case 2, we used a machine that has the Kali Linux OS to emulate the attacker and a machine that has the Ubuntu OS to emulate the victim's machine for the training and deployment phases. This use case has multiple vulnerabilities (four, in this case) and 256 payloads that can be used with any of the vulnerabilities, which creates a complex environment for an RL agent. This creates a good testing use case for using Q-Table and DRL compared with the single vulnerability use case (Section 4.1.1).

Table 2. Group of vulnerabilities.

CVE	Description
CVE-2011-3556 [28]	Unspecified vulnerability in the Java Runtime Environment component in Oracle Java SE JDK and JRE
CVE-2007-2447 [29]	The MS-RPC functionality in smbd
CVE-2004-2687 [30]	distcc 2.x, as used in XCode 1.5 and others
CVE-2012-1823 [31]	sapi/cgi/cgi_main.c in PHP

4.2. RL Training and Deployment: Q-Table

In this subsection, we discuss Q-Table's training and deployment for use cases 1 and 2. The state 'S' of the system refers to the combination of an OS and a certain vulnerability. The Metasploit payloads that are used to exploit the vulnerability and change its state to a compromised OS represent the actions that can be performed 'A'. Table 3 summarizes the states, actions, and RL parameters that exist in the system.

In the following list, we explain how the RL parameters influence the training process:

- Learning rate (α): This controls how much weight the agent gives to new information versus past experiences. A higher value means the agent prioritizes new information, while a lower value emphasizes past experiences.
- Discount factor (γ): This balances the value of immediate rewards (benefits right now) with future rewards (benefits later).

- Exploration rate (ϵ): This represents how often the agent tries random actions instead of the one it thinks is best. A higher value means the agent explores more, and a lower value means it sticks with what it knows works.
- Decay rate: The rate at which ϵ is decayed to favor the exploitation of known actions with a high reward over the exploration of random actions.

During the training phase of the Q-Table on the Apache CouchDB vulnerability, we ran seven trials, each running for 500 episodes of exploitation. After training, a vulnerable Apache CouchDB 3.1.0 machine was used to deploy the trained agent. The agent managed to exploit the vulnerable machine in 8.10 s.

Table 3. RL States, actions, and parameters.

States (S)	Actions (A)	RL Parameters
Vulnerable OS (e.g., Windows with CouchDB) referred to by "*vulnerable_OS*" in Figure 2	Each payload from Metasploit represents an action that can be used to exploit the vulnerable OS	Alpha: The learning rate, Gamma: The discount factor, Epsilon: The exploration rate, The decay rate
Exploited OS, attack succeeded. Referred to by "*exploited_OS*" in Figure 2	Use case 1 has 194 actions (i.e., payloads). Use case 2 has 256 actions.	

Our approach involved training an AI agent using RL utilizing the Metasploit framework. Metasploit offers a vast collection of payloads for exploiting vulnerabilities in different OSs. The RL algorithm trains the agent to pick the most effective payload for the job. It accomplishes this by using Metasploit's RPC API to automate tasks within the framework.

While not all payloads work for every situation, the RL approach helps the agent make smart choices and successfully exploit the vulnerability. Figure 3 shows this process. The following points summarize the training process presented in Figures 2 and 3, given the state of the system (OS and vulnerability):

1. The agent sends a request to get ta payload to exploit the vulnerability from MSFRPC.
2. The agent chooses a certain payload to use. Here is how the agent decides which payload to use: with probability ϵ, the agent selects a random payload to be executed (i.e., an action). On the other hand, with probability $(1 - \epsilon)$, the agent selects the best known payload. This is an ϵ-greedy approach.
3. The agent sends a payload from Metasploit to try and exploit the vulnerability (take action).
4. After using a payload, the agent observes the outcome (new system state) and receives a reward (success or failure signal). This process repeats for a set number of times. As the agent learns, it relies less on random choices (ϵ decreases) and focuses on the most successful payloads. Figure 2 demonstrates the complete training process using Q-Tables.

Our agent considers a successful exploit to be one that opens a reverse shell session. This might not be the goal for every payload in Metasploit, as some might aim for different types of access (like a VNC session). But, for our purposes, a reverse shell signifies success.

To train the agent, we designed a reward system. It gets a high reward (+100) for successfully exploiting a vulnerability with a reverse shell and a penalty (-10) if it fails. These values encourage the agent to prioritize successful exploits and avoid failures.

Since our system only cares about achieving a reverse shell or not, the rewards are kept simple (+100 or -10). This prevents any confusion during training.

The agent's choices are limited to payloads in Metasploit that can specify a local machine and port. To make decisions, the agent relies on its Q-Table, which stores information from past experiences. When facing a new situation (OS and vulnerability), the agent checks its Q-Table to find the best payload for the job.

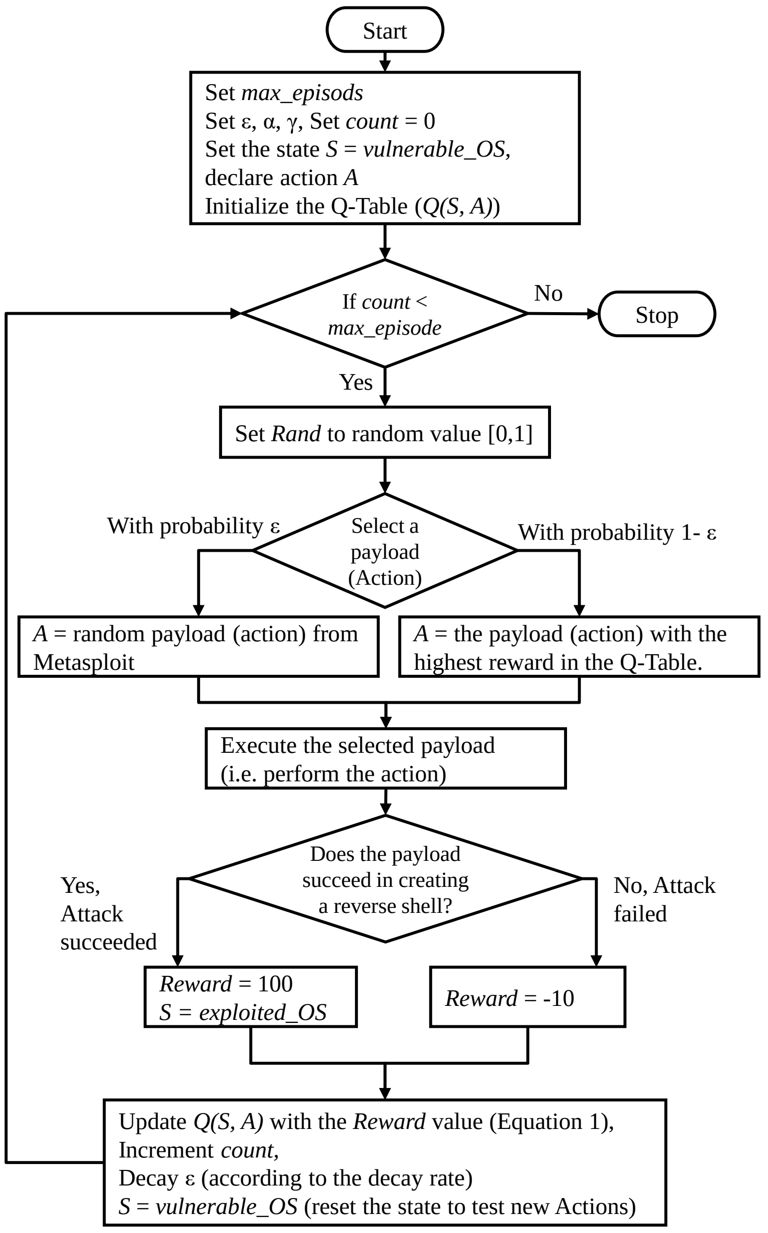

Start
Set *max_episods*
Set ε, α, γ, Set *count* = 0
Set the state *S* = *vulnerable_OS*,
declare action *A*
Initialize the Q-Table (*Q(S, A)*)
If *count* < *max_episode*
No
Stop
Yes
Set *Rand* to random value [0,1]
With probability ε
Select a payload (Action)
With probability 1- ε
A = random payload (action) from Metasploit
A = the payload (action) with the highest reward in the Q-Table.
Execute the selected payload (i.e. perform the action)
Does the payload succeed in creating a reverse shell?
Yes, Attack succeeded
No, Attack failed
Reward = 100
S = *exploited_OS*
Reward = -10
Update *Q(S, A)* with the *Reward* value (Equation 1),
Increment *count*,
Decay ε (according to the decay rate)
S = *vulnerable_OS* (reset the state to test new Actions)

Figure 2. The flowchart of the training process using Q-Tables.

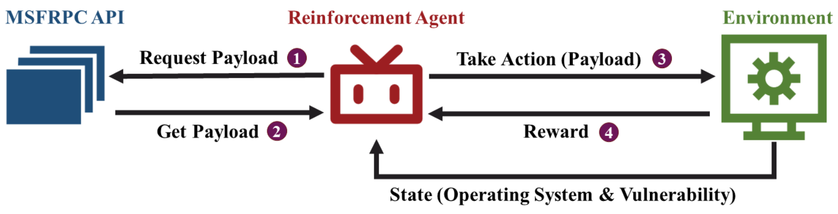
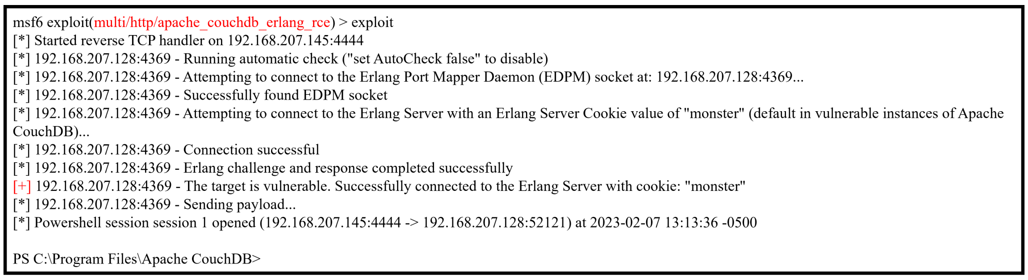


Figure 3. The learning process of the RL agent.

We tested the agent's decision-making results by letting it recommend a payload. The chosen payload was successfully delivered to the target machine and opened a special remote connection (reverse shell session), proving the agent's decision was a good one. This process is illustrated in Figure 4, where the agent picks the payload with the highest reward value. In this case, the payload name can be seen in the first row of Figure 4, which is apache_couchdb_erlang_rce, and is used by MSFRPC to exploit the vulnerability.

The goal of this payload is to open a reverse shell by the attacker with the victim machine. The remaining lines in the figure demonstrate the verbosity of the command, which ends by indicating a shell that connects the attacker with the victim machine.

For the second use case (GV), the same methodology was used, but instead of targeting CouchDB, the vulnerabilities listed in Table 2 were targeted. The state of the system now includes an uncompromised OS with a list of vulnerabilities that can be exploited using certain actions (Metasploit payloads), which may change the state of the system to a compromised OS. While the first use case has one vulnerability and 194 actions, the second use case has four vulnerabilities and 256 actions (i.e., payloads).



```
msf6 exploit(multi/http/apache_couchdb_erlang_rce) > exploit
[*] Started reverse TCP handler on 192.168.207.145:4444
[*] 192.168.207.128:4369 - Running automatic check ("set AutoCheck false" to disable)
[*] 192.168.207.128:4369 - Attempting to connect to the Erlang Port Mapper Daemon (EDPM) socket at: 192.168.207.128:4369...
[*] 192.168.207.128:4369 - Successfully found EDPM socket
[*] 192.168.207.128:4369 - Attempting to connect to the Erlang Server with an Erlang Server Cookie value of "monster" (default in vulnerable instances of Apache
CouchDB)...
[*] 192.168.207.128:4369 - Connection successful
[*] 192.168.207.128:4369 - Erlang challenge and response completed successfully
[+] 192.168.207.128:4369 - The target is vulnerable. Successfully connected to the Erlang Server with cookie: "monster"
[*] 192.168.207.128:4369 - Sending payload...
[*] Powershell session session 1 opened (192.168.207.145:4444 -> 192.168.207.128:52121) at 2023-02-07 13:13:36 -0500

PS C:\Program Files\Apache CouchDB>
```

Figure 4. Successful reverse shell established based on the agent's recommended payload.

4.3. *RL Training and Deployment: DRL*

DRL offers a promising approach for automating penetration testing by enabling agents to learn optimal exploitation strategies within the Metasploit framework. This subsection explores the specifics of DRL training and deployment in this context, focusing on its advantages over Q-learning.

4.3.1. Training Specifics for Metasploit Integration

Here, the DRL agent leverages the Metasploit RPC (MSFRPC) API to interact with the environment. The agent is trained using the DQN algorithm in a simulated environment replicating target systems.

The reward function plays a crucial role during the training process. In this case, a successful reverse shell session established through a chosen payload for a specific OS and vulnerability combination signifies a positive reward, while unsuccessful attempts receive negative rewards. The MSFRPC API provides the agent with available payloads for a given scenario.

4.3.2. DRL Deployment in Metasploit Environment

Deploying a DRL agent trained in simulations within the Metasploit environment required the continuous monitoring of the agent's behavior and logging of its actions, which are essential for evaluating its performance, identifying potential biases, and ensuring responsible use. In this work, we evaluated the performance using the following metrics: success rate and the moving average of the rewards. In addition, DRL has a great advantage over Q-Tables; the deployment (testing) phase is dynamic and achieves continuous learning, which increases the accuracy of the results.

4.3.3. Testing Considerations

After training the agents, the agents are ready for testing. Testing the DRL agent required developing a comprehensive set of test cases covering various scenarios, including different OSs, vulnerabilities, and available payloads, as shown in the next section. Such test cases allow us to evaluate the agent's effectiveness, efficiency, and robustness in diverse situations.

4.3.4. Advantages over Q-Learning

While Q-learning is a popular RL technique, DRL offers several advantages in this specific application:

- **Scalability:** Q-learning requires storing Q-values for all possible state-action pairs, making it impractical for large state and action spaces. DRL, using function approximation techniques like NNs, can efficiently handle complex environments with vast state and action spaces like the one encountered in penetration testing. In the case of GV, adding one extra vulnerability doubles the size of the Q-Table, whereas in DRL, the same DQN can handle this complexity without bloating its size, making it much more scalable.
- **Continuous Learning:** DRL agents can continuously learn and improve their performance over time by interacting with the environment. Q-learning typically requires manual updates to the Q-Table (e.g., new states), making it less suitable for dynamic environments where vulnerabilities and available payloads might change. This justifies the improvement in the accuracy of the system when using DRL over Q-Table, given that the learning continues after the deployment phase in DRL.
- **Generalization:** DRL agents can learn from past experiences and generalize their knowledge to unseen situations because of using NNs. This allows the agents to adapt to different vulnerabilities and OSs. Q-learning, on the other hand, struggles with generalization and requires retraining for each new scenario. In our future work, we plan to test vulnerabilities that were not used in the training phase. This is the ultimate goal of this line of research.

By leveraging the strengths of DRL, this approach paves the way for autonomous penetration testing tools that can learn, adapt, and efficiently navigate complex landscapes within the Metasploit framework.

The two use cases, CouchDB (Section 4.1.1) and GV (Section 4.1.2), were tested using DRL instead of Q-Table. The results are presented and discussed in the next section.

5. Results and Discussion

In this section, we present the results in two subsections. The first subsection presents the results of the vulnerability exploitation of CouchDB using Q-Table and DRL (use case 1). The second subsection presents the results of vulnerability exploitation for GV using Q-Table and DRL (use case 2). Table 4 presents the execution time of the model training and the execution time of the actual vulnerability exploitation. Vulnerability exploitation was conducted using the payload with the highest reward selected after the training.

Table 4. Execution time of model training and vulnerability exploitation.

Use Case	Execution Time (Model Training, Vulnerability Exploitation)
CouchDB using Q-Table	2.5 h, 8.1 s
CouchDB using DRL	2.5 h, 8 s
GV using Q-Table	4.3 h, 8 s
GV using DRL	4.5 h, 8 s

5.1. CouchDB Using Q-Table and DRL

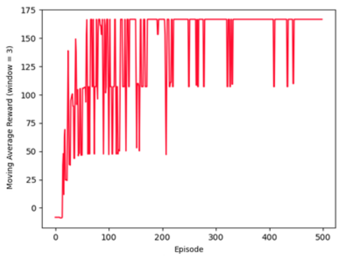
Figure 5 and Table 5 summarize the results for exploiting CouchDB using Q-Table. To assess the performance of the RL agent, we plotted the moving average of Q-values, which provides a smoothed representation of the agent's learning progress over time. A moving average plot for Q-values demonstrates the trend of the agent's Q-value estimates, highlighting how they change as the agent interacts with the environment and learns from its experiences.

Figure 5 shows the results after running the agent for 500 episodes each time. We chose this number of episodes because running for more episodes would not improve the results any further, but it would take much longer and use more computer power.

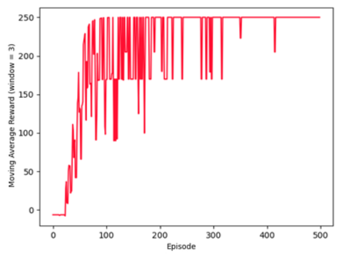
For each run, the success rate (SR) was calculated. This is simply the number of reverse shell sessions successfully established by the agent (x) out of the total number of runs (N), as shown in Equation (3).

$$SR = x/N \tag{3}$$

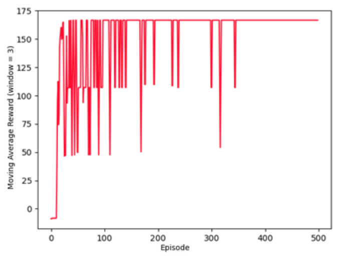
Finding the perfect settings for the agent's learning (α and γ) can be tricky, but they play a big role in how well it performs. Our results (Table 5) show that starting with settings in the middle range (around 0.6) for both how much the agent learns from new things (α) and how often it tries random actions (γ) leads to the best results.



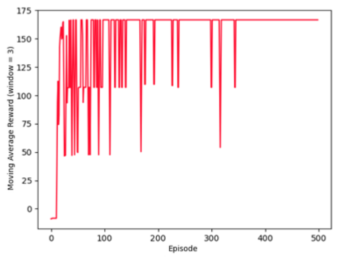
(a) 1, 0.6, 0.4, 0.01, 80.4%



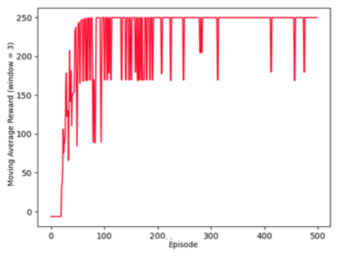
(b) 1, 0.4, 0.6, 0.01, 81.4%



(c) 0.6, 0.6, 0.4 0.01, 88.4%



(d) 1, 0.4, 0.6 0.001, 82.2%



(e) 0.6, 0.4, 0.6, 0.001, 86.2%

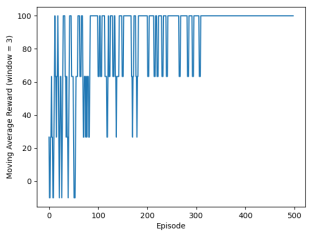
Figure 5. Rewards' moving average for CouchDB using Q-Table in the training phase for 5 scenarios where the values of ϵ, α, γ, rate of decrease, and success rate are shown, respectively, under each figure.

Table 5. Experiment parameters and results for CouchDB exploitation using Q-Table.

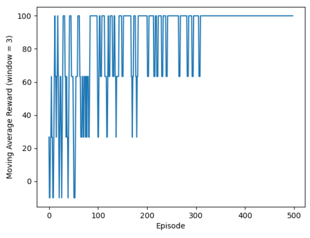
Trial	Epsilon	Alpha	Gamma	Decay Rate	SR %
1	1×10^{1}	6×10^{-1}	4×10^{-1}	1×10^{-2}	80.4
2	1×10^{1}	4×10^{-1}	6×10^{-1}	1×10^{-2}	81.4
3	6×10^{-1}	6×10^{-1}	4×10^{-1}	1×10^{-2}	88.4
4	1×10^{1}	4×10^{-1}	6×10^{-1}	1×10^{-3}	82.2
5	6×10^{-1}	4×10^{-1}	6×10^{-1}	1×10^{-3}	86.2

This is because this balance allows the agent to learn effectively while also exploring new possibilities. We also found that gradually reducing the random actions (exploration) over time leads to better success rates. In short, all our tests were successful after running the training for 500 episodes. This shows that 500 episodes is enough training time for the agent.

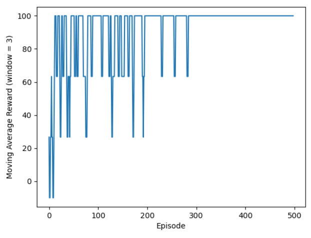
On the other hand, Figure 6 and Table 6 summarize the results for exploiting CouchDB using DRL. Figure 6 demonstrates that the moving average of the rewards has already converged before 500 episodes, similarly to Figure 5. Table 6 shows interesting results for this scenario. The accuracy improved overall compared with when a Q-Table was used. This can be explained by the *continuous learning* characteristic of the DRL agent as it interacts with the environment. By using DRL, a maximum accuracy of 96.6% was achieved, compared with 88.4 using Q-Table.



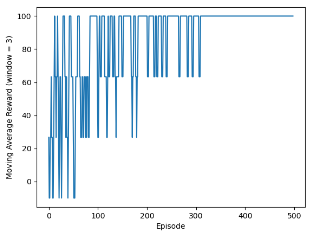
(a) 1.0, 0.6, 0.4, 0.01, 89.2%



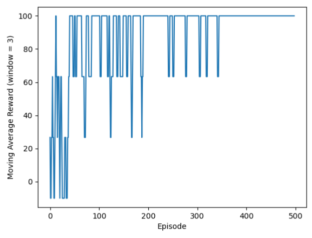
(b) 1, 0.4, 0.6, 0.01, 90.2%



(c) 0.6, 0.6, 0.4, 0.01, 96.6%



(d) 1.0, 0.4, 0.4, 0.01, 91.2%



(e) 0.6, 0.4, 0.6, 0.001, 94.6%

Figure 6. Rewards' moving average for CouchDB using DRL in the training phase for 5 scenarios where the values of ϵ, α, γ, rate of decrease, and success rate are shown, respectively, under each figure.

Table 6. Experiment parameters and results for CouchDB exploitation using DRL.

Trial	Epsilon	Alpha	Gamma	Decay Rate	SR %
1	1×10^{1}	6×10^{-1}	4×10^{-1}	1×10^{-2}	89.2%
2	1×10^{1}	4×10^{-1}	6×10^{-1}	1×10^{-2}	90.2%
3	6×10^{-1}	6×10^{-1}	4×10^{-1}	1×10^{-2}	96.6%
4	1×10^{1}	4×10^{-1}	6×10^{-1}	1×10^{-3}	91.2%
5	6×10^{-1}	4×10^{-1}	6×10^{-1}	1×10^{-3}	94.6%

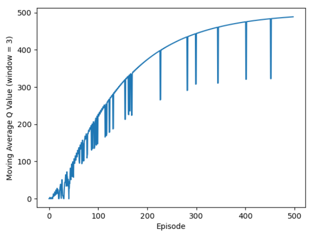
5.2. GV Using Q-Table and DRL

Figure 7 and Table 7 present the results of exploiting GV using Q-Table. Figure 7 demonstrates that the moving average of the reward converges for different scenarios and hyperparameter values of ϵ, α, γ, and the rate of decay. Table 7 demonstrates the success rate for exploiting GV using Q-Table. It is notable that the success rate is much lower than that presented in Table 5. The lower success rate (the best trial achieved 71.2%) can be attributed to considering a larger search space. In this case, the RL agent evaluated four vulnerabilities instead of just one, along with a wider range of actions (256 compared with 194).

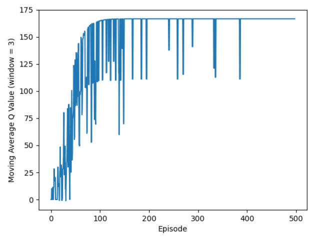
Table 7. Experiment parameters and results for GV exploitation using Q-Table.

Trial	Epsilon	Alpha	Gamma	Decay Rate	SR %
1	7×10^{-1}	1×10^{-1}	8×10^{-1}	1×10^{-2}	69.5
2	1×10^{1}	3×10^{-1}	4×10^{-1}	1×10^{-2}	71.2
3	6×10^{-1}	3×10^{-1}	4×10^{-1}	1×10^{-2}	70.6
4	5×10^{-1}	9×10^{-1}	8×10^{-1}	1×10^{-2}	22.8
5	7×10^{-1}	1×10^{-1}	8×10^{-1}	4×10^{-1}	18.1

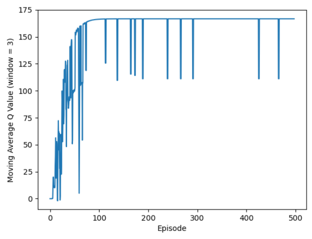
Figure 8 and Table 8 present the results of exploiting GV using DRL. Throughout our experimentation, we conducted five trials with different hyperparameters for the RL agent, and the details of these trials can be found in Table 8, where the best success rate 73.6% was achieved in Figure 8b.



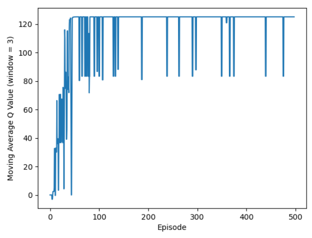
(a) 0.7, 0.1, 0.8, 0.01, 69.5%



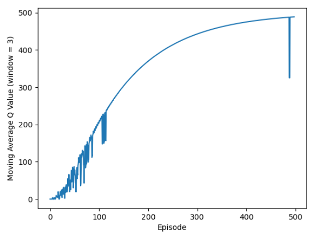
(b) 1, 0.3, 0.4, 0.01, 71.2%,



(c) 0.6, 0.3, 0.4, 0.01, 70.6%



(d) 0.5, 0.9, 0.8, 0.01, 22.8%



(e) 0.7, 0.1, 0.8, 0.40, 18.1%

Figure 7. Rewards' moving average for GV using Q-Table in the training phase for 5 scenarios where the values of ϵ, α, γ, rate of decrease, and success rate are shown, respectively, under each figure.

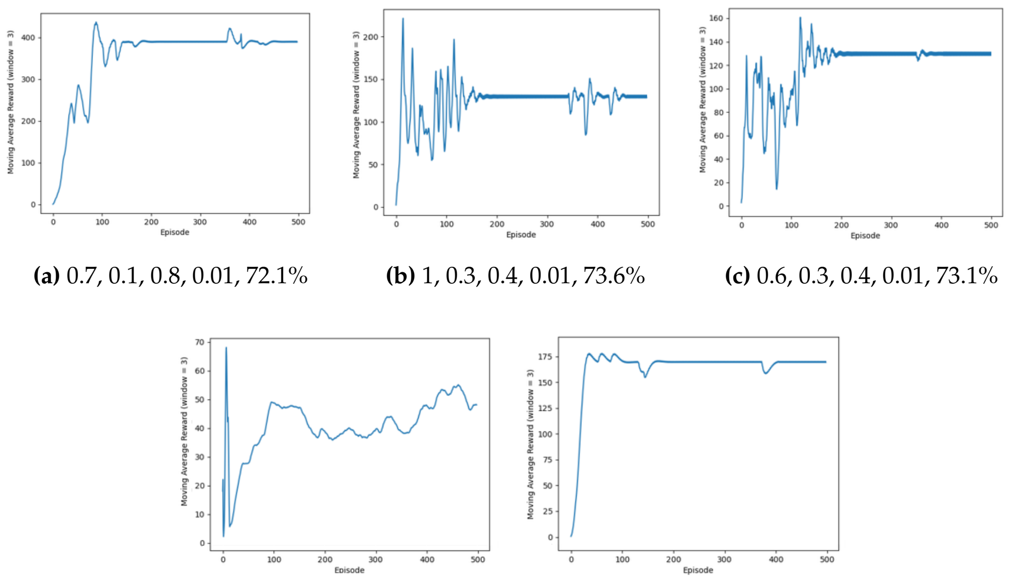
In Figure 8, we present the results of these trials. Figure 8a shows that the agent has achieved an optimal policy with a notable success rate, reaching 72.1% in terms of the number of successful episodes. This trial stands out as the best-performing one due to the well-balanced value of ϵ, allowing for a trade-off between exploration and exploitation. Additionally, the success rate is relatively high, indicating that the agent has effectively learned an optimal policy.

Figure 8b exhibits a high success rate; however, the agent has not yet attained an optimal policy. The value of epsilon encourages the agent to prioritize exploration over exploitation, resulting in ongoing learning without fully converging to an optimal solution. Similarly, in Figure 8c, we observe a similar trend of high success rates but the absence of an optimal policy, even though the value of epsilon balances between exploration and exploitation.

Figure 8d indicates that neither the success rate nor the agent's policy is satisfactory. In this trial, the agent struggles to achieve significant progress in learning and fails to reach an optimal policy. On the other hand, Figure 8e reveals that the agent has reached an optimal policy; however, the success rate is relatively low compared with the other trials.

By analyzing these figures, we can observe the varying outcomes of the different trials, showcasing the impact of different hyperparameters on the agent's learning process and performance.

Similarly to the results presented in Section 5.1, using DRL again achieved better accuracy by having a higher overall success rate than Q-Table. This was noted, as DRL scored a maximum success rate of 73.6% (Table 8) compared with 71.2% (Table 7) for Q-Table. This can be explained again by the *continuous learning* characteristic of the DRL agent as it interacts with the environment.



(a) 0.7, 0.1, 0.8, 0.01, 72.1% **(b)** 1, 0.3, 0.4, 0.01, 73.6% **(c)** 0.6, 0.3, 0.4, 0.01, 73.1%

(d) 0.5, 0.9, 0.8, 0.01, 40.6% **(e)** 0.7, 0.1, 0.8, 0.40, 40%

Figure 8. Rewards' moving average for GV using DRL in the training phase for 5 scenarios where the values of ϵ, α, γ, rate of decrease, and success rate are shown, respectively, under each figure.

Table 8. Experiment parameters and results of GV exploitation using DRL.

Trial	Epsilon	Alpha	Gamma	Decay Rate	SR %
1	7×10^{-1}	1×10^{-1}	8×10^{-1}	1×10^{-2}	72.1
2	1×10^{-1}	3×10^{-1}	4×10^{-1}	1×10^{-2}	73.6
3	6×10^{-1}	3×10^{-1}	4×10^{-1}	1×10^{-2}	73.1
4	5×10^{-1}	9×10^{-1}	8×10^{-1}	1×10^{-2}	40.6
5	7×10^{-1}	1×10^{-1}	8×10^{-1}	4×10^{-1}	40.0

6. Conclusions and Future Work

In this paper, we utilized RL to perform vulnerability exploitation. We built two RL agents: the first one uses Q-Table and the second one uses DRL. The state of the RL agent is identified by the OS and vulnerability under investigation. The actions of the RL agent are identified by the Metasploit payloads that can be used to exploit the given vulnerabilities. An exploitation is considered successful if the payload was able to create a reverse shell with the attacker. Both RL agents (i.e., the one using Q-Table and the one using DRL) were trained and tested on two use cases. The first use case has one vulnerability, while the second use case has four vulnerabilities.

Our results demonstrate that using DRL outperformed Q-Table in the two use cases. In the first use case, DRL had a maximum success rate of 96.6%, while Q-Table had 88.4%. For the second use case, DRL had a maximum success rate of 73.6%, while Q-Table had 71.2%. The DRL agent's higher success rate stems from its ability to continuously learn and adapt during deployment, unlike the static Q-Table approach.

This research introduces a new way to automate tasks during pentesting, making them faster and less expensive. Our method uses an AI agent trained with RL to take advantage of the Metasploit framework. By automating the exploitation process, our approach frees up security specialists to focus on other important tasks. This can significantly reduce the time and resources needed to identify and address vulnerabilities in computer systems.

For future work, we plan to build a general agent that can exploit new vulnerabilities on which it was not trained. To achieve this, we are considering adding more information to the agent's decision-making process like the Common Weakness Enumeration (CWE). This information could be used to make better decisions and exploit new scenarios and vulnerabilities.

Author Contributions: Conceptualization, A.A., L.A.-A., Z.S., K.M.A.Y. and B.J.M.; methodology, A.A., L.A.-A., Z.S., K.M.A.Y. and B.J.M.; software, L.A.-A., Z.S. and A.A.S.; validation, L.A.-A., Z.S. and A.A.S.; formal analysis, A.A., L.A.-A., K.M.A.Y., B.J.M. and Z.S.; investigation, A.A., L.A.-A., K.M.A.Y., B.J.M. and Z.S.; data curation, A.A., L.A.-A., K.M.A.Y., B.J.M. and Z.S.; writing—original draft preparation, A.A., L.A.-A., K.M.A.Y., B.J.M. and Z.S.; writing—review and editing, A.A., K.M.A.Y. and B.J.M.; visualization, A.A., L.A.-A., K.M.A.Y. and B.J.M.; supervision, A.A.; project administration, A.A.; funding acquisition, A.A. All authors have read and agreed to the published version of the manuscript.

Funding: The work in this paper was supported, in part, by the Open Access Program from the American University of Sharjah.

Institutional Review Board Statement: This paper represents the opinions of the author(s) and does not mean to represent the position or opinions of the American University of Sharjah.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

IT	Information Technology
OT	Operation Technology
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
ML	Machine Learning
DQN	Deep Q-Network
R	Reward
NIST	National Institute of Standards and Technology
DARPA	Defense Advanced Research Projects Agency
CTF	Capture The Flag
GV	Group of Vulnerabilities
SR	Success Rate
CWE	Common Weakness Enumeration

References

1. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [CrossRef] [PubMed]
2. Perera, S.; Jin, X.; Maurushat, A.; Opoku, D.G.J. Factors Affecting Reputational Damage to Organisations Due to Cyberattacks. *Informatics* **2022**, *9*, 28. [CrossRef]
3. Perkal, Y. Is Your Vulnerability Scanner Giving You Reliable Results? 2022. Available online: https://securityboulevard.com/2022/10/is-your-vulnerability-scanner-giving-you-reliable-results/ (accessed on 12 October 2024).
4. Metasploit. Metasploit-Framework. 2023. Available online: https://www.metasploit.com/ (accessed on 22 June 2024).
5. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
6. Dong, S.; Wang, P.; Abbas, K. A survey on deep learning and its applications. *Comput. Sci. Rev.* **2021**, *40*, 100379. [CrossRef]
7. Shu, L.; Dong, S.; Su, H.; Huang, J. Android malware detection methods based on convolutional neural network: A survey. *IEEE Trans. Emerg. Top. Comput. Intell.* **2023**, *7*, 1330–1350. [CrossRef]
8. Nieles, M.; Dempsey, K.; Pillitteri, V.Y. NIST Special Publication 800-12. *DRAFT Revis.* **2017**, *1*. [CrossRef]
9. Kennedy, D.; O'Gorman, J.; Kearns, D.; Aharoni, M. *Metasploit: The Penetration Tester's Guide*; No Starch Press: San Francisco, CA, USA, 2011.
10. NIST. NIST Special Publication 800-30 Revision 1-Guide for Conducting Risk Assessments. 2012. [CrossRef]
11. DARPA. Cyber Grand Challenge (CGC). 2016. Available online: https://www.darpa.mil/program/cyber-grand-challenge (accessed on 25 February 2024).
12. Maddala, S.; Patil, S. Agentless automation model for post exploitation penetration testing. In *Proceedings of the Intelligent Computing, Information and Control Systems: ICICCS 2019, Madurai, India, 15–17 May 2019*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 529–539.

13. Chaudhary, S.; O'Brien, A.; Xu, S. Automated Post-Breach Penetration Testing through Reinforcement Learning. In Proceedings of the 2020 IEEE Conference on Communications and Network Security (CNS), Avignon, France, 29 June–1 July 2020; pp. 1–2. [CrossRef]
14. Benito, R.; Shaffer, A.; Singh, G. An Automated Post-Exploitation Model for Cyber Red Teaming. In Proceedings of the International Conference on Cyber Warfare and Security, Towson, MD, USA, 9–10 March 2023; Volume 18, pp. 25–34.
15. Maeda, R.; Mimura, M. Automating post-exploitation with deep reinforcement learning. *Comput. Secur.* **2021**, *100*, 102108. [CrossRef]
16. Hu, Z.; Beuran, R.; Tan, Y. Automated Penetration Testing Using Deep Reinforcement Learning. In Proceedings of the 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS & PW), Genoa, Italy, 7–11 September 2020; pp. 2–10. [CrossRef]
17. Schwartz, J.; Kurniawati, H. Autonomous penetration testing using reinforcement learning. *arXiv* **2019**, arXiv:1905.05965.
18. Ghanem, M.C.; Chen, T.M. Reinforcement Learning for Intelligent Penetration Testing. In Proceedings of the 2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), London, UK, 30–31 October 2018; pp. 185–192. [CrossRef]
19. Ghanem, M.C.; Chen, T.M. Reinforcement learning for efficient network penetration testing. *Information* **2019**, *11*, 6. [CrossRef]
20. Ghanem, M.C.; Chen, T.M.; Nepomuceno, E.G. Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks. *J. Intell. Inf. Syst.* **2023**, *60*, 281–303. [CrossRef]
21. Zennaro, F.M.; Erdődi, 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**, *17*, 441–457. [CrossRef]
22. Erdődi, L.; Sommervoll, A.A.; Zennaro, F.M. Simulating SQL injection vulnerability exploitation using Q-learning reinforcement learning agents. *J. Inf. Secur. Appl.* **2021**, *61*, 102903. [CrossRef]
23. Tran, K.; Standen, M.; Kim, J.; Bowman, D.; Richer, T.; Akella, A.; Lin, C.T. Cascaded Reinforcement Learning Agents for Large Action Spaces in Autonomous Penetration Testing. *Appl. Sci.* **2022**, *12*, 1265. [CrossRef]
24. Yi, J.; Liu, X. Deep Reinforcement Learning for Intelligent Penetration Testing Path Design. *Appl. Sci.* **2023**, *13*, 9467. [CrossRef]
25. AlMajali, A.; Al-Abed, L.; Mutleq, R.; Samamah, Z.; Shhadeh, A.A.; Mohd, B.J.; Yousef, K.M.A. Vulnerability Exploitation Using Reinforcement Learning. In Proceedings of the 2023 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), Amman, Jordan, 22–24 May 2023; pp. 281–286.
26. Apache. Apache CouchDB. 2023. Available online: https://couchdb.apache.org/ (accessed on 10 February 2023).
27. Justicz, M.; Touzet, J. CouchDB Vulnerability. 2023. Available online: https://www.rapid7.com/db/modules/exploit/linux/http/apache_couchdb_cmd_exec/ (accessed on 10 February 2023).
28. NIST. CVE-2011-3556. 2011. Available online: https://nvd.nist.gov/vuln/detail/CVE-2011-3556 (accessed on 6 June 2024).
29. MITRE. CVE-2007-2447. 2007. Available online: https://cve.mitre.org/cgi-bin/cvename.cgi?name=2007-2447 (accessed on 6 June 2024).
30. NIST. CVE-2004-2687. 2004. Available online: https://nvd.nist.gov/vuln/detail/CVE-2004-2687 (accessed on 6 June 2024).
31. NIST. CVE-2012-1823. 2012. Available online: https://nvd.nist.gov/vuln/detail/cve-2012-1823 (accessed on 6 June 2024).