2023 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT) | 979-8-3503-2405-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/JEEIT58638.2023.10185700

Vulnerability Exploitation Using Reinforcement Learning

Anas AlMajali
Department of Computer Engineering
Faculty of Engineering
The Hashemite University
Zarqa, Jordan
almajali@hu.edu.jo

Loiy Al-Abed
Department of Computer Engineering
Faculty of Engineering
The Hashemite University
Zarqa, Jordan
1830887@std.hu.edu.jo

Ruba Mutleq
Department of Computer Engineering
Faculty of Engineering
The Hashemite University
Zarqa, Jordan
1834219@std.hu.edu.jo

Zaid Samamah
Department of Computer Engineering
Faculty of Engineering
The Hashemite University
Zarqa, Jordan
1933504@std.hu.edu.jo

Anas Abu Shhadeh
Department of Computer Engineering
Faculty of Engineering
The Hashemite University
Zarqa, Jordan
2030255@std.hu.edu.jo

Bassam J. Mohd
Department of Computer Engineering
Faculty of Engineering
The Hashemite University
Zarqa, Jordan
bassam@hu.edu.jo

Khalil M. Ahmad Yousef
Department of Computer Engineering
Faculty of Engineering
The Hashemite University
Zarqa, Jordan
khalil@hu.edu.jo

***Abstract*—Our main goal is to create a reinforcement agent that is capable of exploiting a particular vulnerability. Hiring a penetration tester or doing manual exploitation can be expensive and time-consuming, thus such a process needs to be intelligent and automated. There are many tools out there that perform auto-exploitation, like Metasploit Pro. But the problem with such tools is that they require significant execution time and resources because they are based on trying every possible payload and checking if it works or not. In this work, we created a reinforcement agent and configured it to exploit a certain vulnerability. After the agent completes the training phase, it stores payloads with their corresponding reward values in a Q-Table. When the agent faces a state that is a combination of a target operating system and a certain vulnerability, it knows what options to set to perform exploitation by looking at its Q-Table. The proposed methodology was tested on remote code execution vulnerability in CouchDB version 3.1.0. After the training phase was completed, deployment was tested on three different systems in which the main goal of the attacker (establishing a reverse shell) was achieved using the payloads with the highest rank in the Q-Table in 8.26 seconds (average).**

***Index Terms*—Reinforcement learning, penetration testing, cybersecurity**

I. Introduction

Reinforcement Learning (RL) is a subfield of Machine Learning (ML) that deals with the problem of an agent learning to interact with its environment in order to maximize a reward signal through trial and error. The concept of RL has been around for several decades, but it was not until the late 20th century that researchers started to develop mathematical frameworks for the problem. Early work on RL focused on the development of algorithms for solving Markov Decision Processes (MDPs), which provide a mathematical framework for modeling sequential decision-making problems [1].

RL gained widespread attention in the late 1990s, when researchers started to apply RL techniques to a wide range of problems, including control and optimization, gaming, and autonomous agents. In recent years, RL has become an active area of research, with many new algorithms and techniques being developed. The development of deep RL, which combines deep learning with RL, has allowed for the application of RL to complex and real-world problems, such as game playing and robotics [2]. RL is often referred to as Q or Deep-Q learning. Q stands for quality, which represents how useful a given action is in gaining some future reward.

The basic idea behind RL is to train an agent to maximize a reward signal by taking actions that lead to high rewards and avoiding actions that lead to low rewards. The learning process involves the agent receiving feedback in the form of rewards for its actions and updating its decision-making strategy based on this feedback. This process continues until the agent converges to an optimal policy, which is a mapping from states to actions that maximizes the expected cumulative reward over time. RL has found applications in a variety of domains, including robotics, video games, and finance, and has been used to achieve superhuman performance in complex games such as chess and Go. The significance of RL lies in its ability to handle complex and dynamic environments, where a model-based approach may not be feasible. RL algorithms can learn from their experiences, continually adapting to new scenarios, and improving their performance over time. This makes it an essential tool for solving problems where the solution is unknown and requires an agent to learn by interacting with its environment [3].

Penetration testing, also referred to as pen-testing, is a method of evaluating the security of computer systems or networks by simulating an attack by a malicious individual. This security assessment technique has been used for several decades to identify potential vulnerabilities that an attacker could exploit. The goal of penetration testing is to provide organizations with actionable information about their security posture, enabling them to identify and prioritize areas of risk and improve their overall security [4].

The history of penetration testing dates back to the early days of computer security when researchers first started to investigate the security of computer systems. Over the years, the field of penetration testing has evolved and matured, becoming a well-established and widely used technique for evaluating the security of computer systems and networks. Today, organizations across a wide range of industries and of varying sizes rely on penetration testing to assess the security of their critical systems, including networks, websites, applications, and other IT assets [5].

Automated penetration testing is an automated way of detecting vulnerabilities using penetration testing tools. It saves time compared to manual testing and uses advanced algorithms, machine learning, and AI to scan systems for vulnerabilities. The results are analyzed to identify weaknesses and make security recommendations such as applying patches, upgrading software, or modifying configurations. Automated penetration testing is vital as it offers organizations an efficient, scalable, and cost-effective way to assess system security. Automation enables regular security assessments and real-time monitoring for maintaining a secure environment [6].

The key to handling false positives is doing penetration testing or in other words, exploiting the reported vulnerabilities [7]. However, hiring a penetration tester or conducting manual exploitation can be a costly and time-consuming process, making automation desirable. There are various tools available for automating exploitation, however, many of them are resource-intensive and lack customization options.

In response to the limitations and challenges associated with traditional methods of penetration testing and manual exploitation, we developed a reinforcement learning-based agent for the purpose of automating the exploitation process. The agent is designed to be trained on various vulnerabilities and operating systems. Thereby acquiring the ability to make informed decisions and carry out exploitation tasks with increased efficiency and accuracy. The research work presented in this paper is a part of a larger scope project aimed at developing a general agent that is capable of exploiting any/general task and making the appropriate decision. By presenting the algorithm and process to train the RL agent for specific tasks, we demonstrate an important stepping stone towards constructing the general agent.

We developed an RL agent that leverages the Q-learning algorithm to exploit a specified vulnerability. The agent operates by evaluating payloads from the Metasploit framework and determining their effectiveness based on the target operating system and vulnerability [8]. Successful exploitation is determined by the establishment of a reverse shell session following payload execution.

Our primary contribution in this paper is the development of a reinforcement learning-based agent that is capable of performing exploitation tasks efficiently. By leveraging the capabilities of the Metasploit framework and its Remote Procedure Call (RPC) API, the agent is able to automate the exploitation process and check if the identified vulnerability is exploitable [9]. As a result, our approach enhances the efficiency of the security assessment process by reducing the time and resources required to identify and exploit target vulnerabilities. This not only has the potential to improve the overall effectiveness of security assessments, but also offers a novel approach to the challenge of vulnerability exploitation. The use of RL offered a flexible and adaptable solution that could continuously improve its exploitation strategy over time, providing a more cost-effective alternative to traditional methods of penetration testing and manual exploitation.

II. Related Work

Previous research on using machine learning for vulnerability assessment and penetration testing has focused on either the post-exploitation phase or optimizing attack paths. A summary of some of the related studies is provided below.

Chaudhary et al. [10] automated the process of penetration testing to identify and exploit vulnerabilities in digital assets by applying machine learning techniques. Specifically, the authors proposed using RL. The agent is trained by providing it with an environment in which it can explore a compromised network and locate sensitive files. The authors' goal was to train the agent in multiple different environments so that the method could be generalized and applicable. The authors also suggest that further research may involve training the agent for further lateral exploration and exploitation within the system.

Maeda and Mimura [11] proposed combining deep RL with a tool called PowerShell Empire, which is a post-exploitation framework, to create agents that can choose actions based on the state of the system they are attacking. The agents are trained using three different RL models: A2C, Q-Learning, and SARSA. The results of the training phase show that the A2C model was the most efficient at gaining rewards. The trained agents were then tested in a different network domain, and the results showed that the agent trained with the A2C model was able to obtain administrative privileges on the domain controller.

Ghanem and Chen [12] proposed improving penetrating testing using RL by actively attempting to exploit vulnerabilities of digital assets. The authors proposed using RL to model the penetration testing process as a Partially Observed Markov Decision Process (POMDP) and solve it using an external POMDP solver with different algorithms. The results of the proposed method support the hypothesis that RL can enhance penetration testing beyond the capabilities of human experts. This approach was limited to the planning phase of penetration testing and did not cover the entire process.

Niculae et al. [13] discussed the use of RL algorithms to automate the process of penetration testing in which attacks were simulated in a computer system in order to identify vulnerabilities. The authors compared several different RL algorithms, including Q-learning, Extended Classifier Systems, and Deep Q-Networks, to check which one was most effective at finding the optimal strategy for an attacker trying to compromise a network. They measured the effectiveness of each algorithm in terms of the speed and stealthiness of the simulated attack. The results showed that Q-learning outperforms human performance, while Extended Classifier Systems performed worse than humans, but were more stable. Deep Q-Networks, on the other hand, had slow convergence and were unable to achieve comparable performance. The authors also found that all of the machine learning approaches outperformed fixed-strategy attackers.

Hu et al. [14] used deep RL to automate the process of penetration testing, which is typically done manually by ethical hackers (also known as ”pen-testers”). The framework used by the authors has two stages. In the first stage, the Shodan search engine was used to collect relevant server data and build a realistic network topology, and then the MulVAL tool was used to generate an attack tree for that topology. Traditional search algorithms were then used to find all possible attack paths in the tree and build a matrix representation needed by deep RL algorithms. In the second stage, the Deep Q-Learning Network (DQN) method was used to discover the most easily exploitable attack path from the candidates. The approach was evaluated by generating thousands of input scenarios, and the DQN was able to find the optimal path with an accuracy of 0.86, while also providing valid solutions in the other cases.

Schwartz and Kurniawati [15] discussed the use of AI, specifically model-free RL, to automate the process of penetration testing (pen-testing). Pentesting involves simulating a controlled attack on a computer system to assess its security, but it requires skilled practitioners and there is a shortage of such professionals. Automating the pentesting process using AI techniques, such as model-free RL, could help address this shortage. In this study, the researchers built a fast, low compute simulator to train and test autonomous pentesting agents. They used the simulator to test the standard Q-learning algorithm with both tabular and neural network-based implementations. The results showed that both implementations were able to find optimal attack paths for different network topologies and sizes without a model of action behavior. However, the algorithms were only practical for smaller networks and numbers of actions. The researchers argued that further work is needed to develop scalable RL algorithms and test them in larger, higher fidelity environments.

In contrast to previous research efforts that have focused on other stages of the security assessment cycle, such as post-exploitation, our approach seeks to leverage RL in the pre-exploitation stage. Specifically, we trained the RL agent to determine which payload from the Metasploit framework is able to establish a reverse shell session for a given operating system and vulnerability. This focus on the pre-exploitation stage offers a novel approach to the challenge of vulnerability exploitation and has the potential to improve the efficiency and accuracy of security assessments. By automating the selection of the exploitation payload, our approach checks if the positively (true or false) identified vulnerability is exploitable.

III. Methodology

In this section, we present the main methodology used in addition to the experimental setup.

A. RL Training and Deployment

The implementation of our reinforcement learning based agent approach was executed within the Metasploit framework, which offers a comprehensive library of payloads for exploiting vulnerabilities in various operating systems. The RL algorithm is utilized to train the agent to identify the most appropriate payload for a given operating system and vulnerability. This is achieved by utilizing Metasploit's RPC API to enable the agent to automate various tasks within the framework. Although not all payloads may prove effective for a particular operating system, the RL approach was instrumental in enabling the agent to make informed decisions and successfully exploit the target vulnerability. This can be illustrated in Fig. 1; where the RL agent sends a Metasploit payload following the exploitation of the given vulnerability (i.e. performs an action). Our agent follows an epsilon greedy strategy, which involves selecting a random action with probability epsilon (a value between 0 and 1) and the best-known action with probability *1 - epsilon*. In our case, the actions are payloads provided by the MSFRPC API. After selecting an action, the agent applies it to the environment and observes the resulting state, which includes information about the operating system and the vulnerability. The environment provides a reward to the agent, which indicates the success or failure of the exploitation attempt. The agent repeats this process for a certain number of iterations, gradually decreasing epsilon after each iteration to focus on exploiting the best-known actions.

Additionally, our agent's criterion for evaluating the effectiveness of a payload is based on the successful establishment of a reverse shell session. While this outcome may not be applicable for all payloads (e.g. some may aim to establish a VNC session), the limitation of our agent is that it assesses success solely based on the establishment of a reverse shell. A reward system has been implemented where the agent receives a reward of +100 in the event of a successful exploitation, and a reward of -10 if the reverse shell session is not established. Opting for reward values of +100 for successful exploitation and -10 for unsuccessful exploitation leads to a situation where

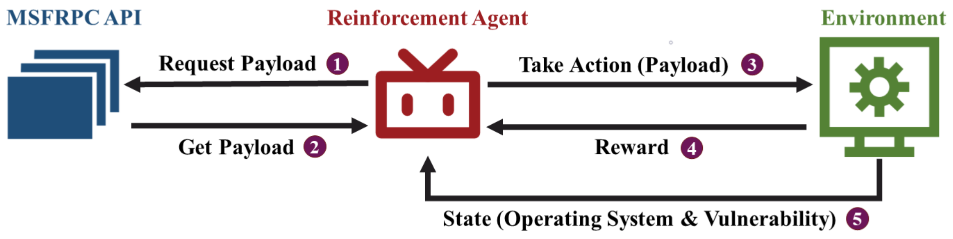


Fig. 1. The work flow of the reinforcement agent. MSFRPC stands for Metasploit Framework Remote Procedure Call.

the agent is strongly motivated to pursue positive rewards and avoid negative ones. This often results in the agent displaying more conservative behavior and being less likely to take risks that could potentially yield negative outcomes. Given that the range of possible rewards in the problem domain is limited to successful and unsuccessful exploitation, it is advisable to select reward values that are consistent with this range. Doing so helps prevent any biases or inconsistencies from being introduced into the learning process.

Examples of payloads that work, but do not open a reverse shell session can be observed in Table I.

TABLE I
PAYLOADS THAT DID NOT OPEN A REVERSE SHELL SESSION

Payload
cmd/windows/powershell/vncinject/reverse_tcp
cmd/windows/powershell/vncinject/reverse_tcp_allports
cmd/windows/powershell/vncinject/reverse_tcp_dns
cmd/windows/powershell/vncinject/reverse_tcp_rc4_dns
cmd/windows/powershell/vncinject/reverse_tcp_uuid
cmd/windows/powershell/x64/vncinject/reverse_tcp

In our reinforcement learning-based approach, the states of the environment are represented as a combination of an operating system and a vulnerability. The agent's actions are limited to those payloads in the Metasploit framework that have parameters for local host and local port. The agent's decision-making process is based on the information stored in its Q-table, which contains the history of its previous experiences. When the agent is presented with a particular state (i.e., a specific operating system and vulnerability), it consults its Q-table to determine the most appropriate payload to use for exploiting the vulnerability.

The efficiency of the reinforcement agent's decision was assessed through the practical application of its recommended payload. The payload was successfully delivered to the target machine and established a reverse shell session, providing validation of the agent's decision. This can be observed in Fig. 2.

B. Experimental Setup

In our study, we chose to focus on a specific vulnerability in Apache CouchDB [16] Version 3.1.0, which is vulnerable to remote code execution [17]. This vulnerability presents a significant security risk and is of particular interest due to its potential for widespread exploitation. Tables II outlines the technical specifications of the attacker and victim machines in the training phase of the reinforcement agent. On the other hand, Tables III, IV, and V outline the technical specifications of the attacker and victim machines in the deployment phase.

TABLE II
SIMULATION 1 SPECIFICATIONS FOR THE TRAINING PHASE

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

TABLE III
SIMULATION 2 SPECIFICATIONS FOR THE DEPLOYMENT PHASE

Attacker (Virtual Machine)	Victim (Host Machine)
4 GB of RAM DDR4	4 GB of RAM DDR4
1 CPU, 2 Cores (Intel Core i7 10th gen) 2.2GHz	1 CPU, 2 Cores (Intel Core i7 10th gen) 2.2GHz
Kali Linux 64-bit	Windows 10 64-bit

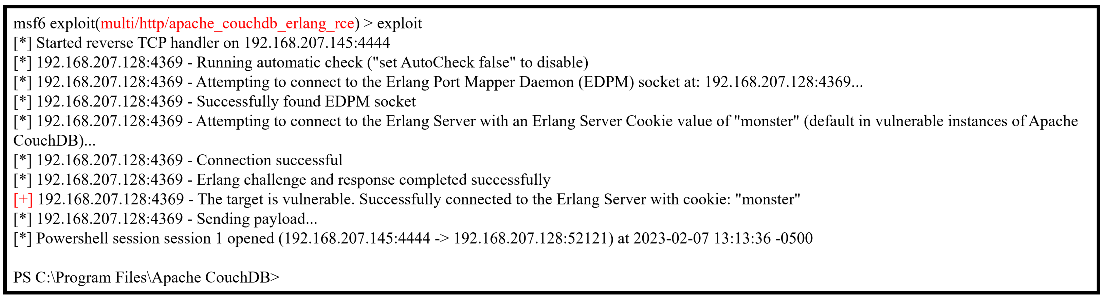
TABLE IV
SIMULATION 3 SPECIFICATIONS FOR THE DEPLOYMENT PHASE

Attacker (Virtual Machine)	Victim (Virtual Machine)
2 GB of RAM DDR4	2 GB of RAM DDR4
1 CPU, 2 Cores (AMD Ryzen 7 4th gen) 2.90GHz	1 CPU, 1 Core (AMD Ryzen 7 4th gen) 2.90GHz
Kali Linux 64-bit	Windows 8 64-bit

TABLE V
SIMULATION 4 SPECIFICATIONS FOR THE DEPLOYMENT PHASE

Attacker (Virtual Machine)	Victim (Virtual Machine)
2 GB of RAM DDR4	8 GB of RAM DDR4
1 CPU, 2 Cores (AMD Ryzen 7 4th gen) 2.90GHz	1 CPU, 2 Cores (AMD Ryzen 7 4th gen) 2.90GHz
Kali Linux 64-bit	Windows 11 64-bit

During the training phase, the agent required an average of 2.5 hours to complete 500 episodes of exploitation out of 7 trials. Subsequently, the agent was deployed on multiple machines that are vulnerable to Apache CouchDB 3.1.0 (Tables III). It is observed that it took mere seconds for the agent to exploit these machines as can be seen in Table VI. Note: all these simulations were carried out using the



```
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>
```

Fig. 2. Successful manual exploitation based on the agent's decision

following payload which the reinforcement agent suggested: cmd/windows/powershell/x64/powershell_reverse_tcp

TABLE VI
EXECUTION TIME FOR EACH SIMULATION

Simulation number	**Execution time**
Simulation 2	7.99 seconds
Simulation 3	8.70 seconds
Simulation 4	8.10 seconds

IV. Results and Discussion

In Fig. 3 the consistency in the number of episodes depicted in each figure is maintained at 500. This was a deliberate choice as a higher number of episodes would result in an increased consumption of time and computational resources with no added value as the moving average of the rewards converges. α determines the extent to which new information should override old information, i.e. the learning rate. A high α value means that new information is given more weight, while a low α value means that old information is given more weight. γ (also known as the discount factor) is used to balance immediate and future rewards. ϵ typically refers to the exploration rate, which represents the probability of an agent taking a random action instead of the action with the highest expected reward. A high value of epsilon represents a high degree of exploration in the agent's behavior. On the other hand, a low value of epsilon represents a high degree of exploitation in the agent's behavior For each run, we compute the success rate which is the number of episodes that resulted in reverse shell session divided by the total number of episodes (500).

The values of α and γ have a significant impact on the performance of an RL algorithm, and finding the optimal values for these parameters can be challenging. Results (in Table VII) show that starting the training with medium range values (i.e. 0.6) for parameter ϵ and α produces best success rates because it balances learning and randomness during training. Additionally, moving slightly faster from randomness (i.e. exploration) to exploitation provides better success rate. In summary, the aforementioned findings demonstrate that all the iterations converged and yielded successful outcomes. This indicates that running the training phase for 500 episodes is sufficient for the training stage.

TABLE VII
EXPERIMENTS PARAMETERS AND RESULTS SUMMARY

Trial	ϵ	α	γ	Rate of decay	Success rate
a	1.0	0.6	0.4	0.01	80.4%
b	1.0	0.4	0.6	0.01	81.4%
c	0.6	0.6	0.4	0.01	88.4%
d	1.0	0.4	0.6	0.001	82.2%
e	0.6	0.4	0.6	0.001	86.2%

The prioritization of exploitation of the best action by an agent may prompt inquiry. However, it is crucial to note that there are instances where a payload may not succeed due to various factors, such as changes in network conditions between attempts, which can impede payload delivery to the target machine. Additionally, certain payloads utilize a staged delivery mechanism, and failure of any part of the payload to deliver correctly may result in the entire payload's failure to execute. Therefore, it is essential for the RL agent to consistently select the action with the highest likelihood of successful exploitation, which is likely to result in a high reward. By consistently selecting the best action, the RL agent maximizes its overall reward, despite the likelihood of payload failure.

V. Conclusion and Future Work

Our primary contribution in this area of research is the development of a reinforcement learning-based agent that is capable of performing exploitation tasks in a more efficient and cost-effective manner. By leveraging the capabilities of the Metasploit framework and its RPC API, the agent is able to automate the exploitation process. As a result, our approach enhances the efficiency of the security assessment process by reducing the time and resources required to identify and exploit target vulnerabilities. This not only has the potential to improve the overall effectiveness of security assessments but also offers a novel approach to the challenge of vulnerability exploitation.

Our solution uses RL to improve the efficiency of security assessments by automatic exploitation as a verification that a certain vulnerability is indeed exploitable. RL algorithms are able to adapt and learn from the behavior of the system or network being assessed. We observed that starting the training with medium range values (i.e. 0.6) for parameter ϵ

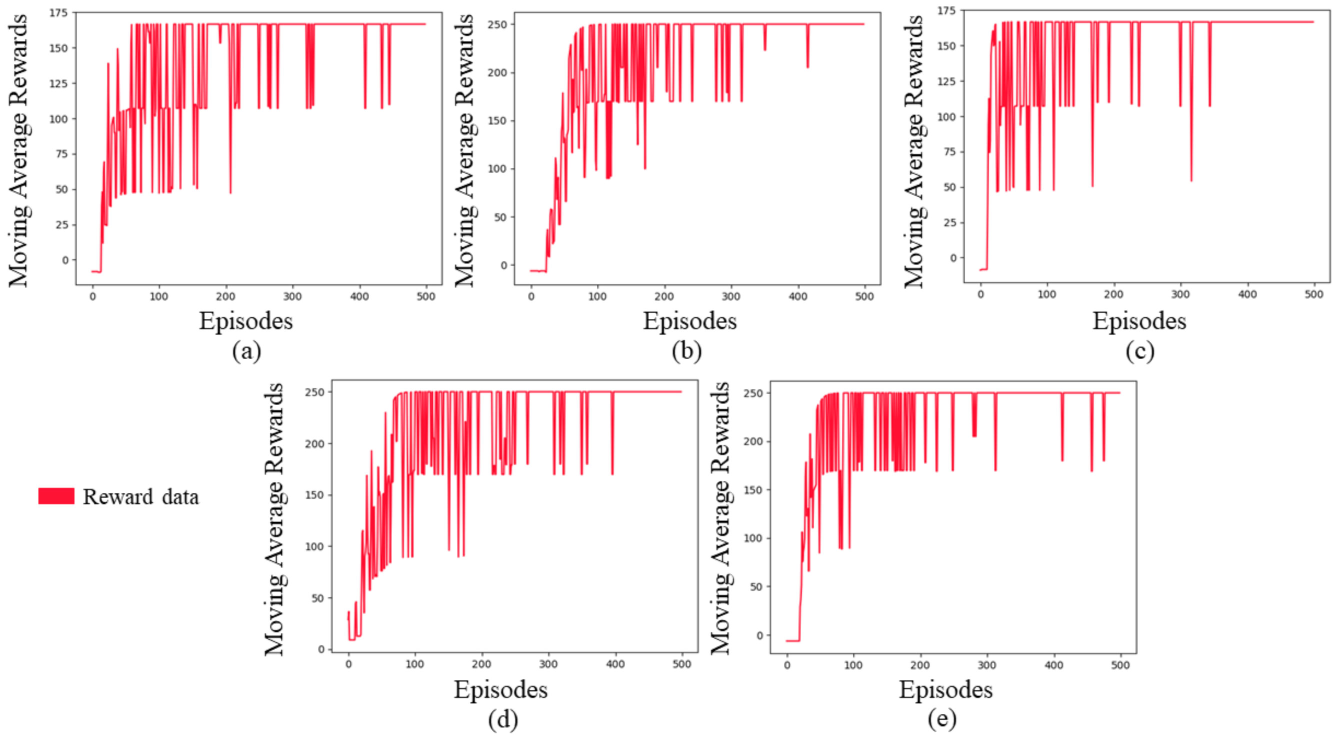


Fig. 3. This figure represents the moving average in the learning phase in five trials: (a) $\epsilon = 1, \alpha = 0.6, \gamma = 0.4$, rate of decrease = 0.01. (b) $\epsilon = 1, \alpha = 0.4, \gamma = 0.6$, rate of decrease = 0.01. (c) $\epsilon = 0.6, \alpha = 0.6, \gamma = 0.4$, rate of decrease = 0.01. (d) $\epsilon = 1, \alpha = 0.4, \gamma = 0.6$, rate of decrease = 0.001 (e) $\epsilon = 0.6, \alpha = 0.4, \gamma = 0.6$, rate of decrease = 0.001. Success rates for figures (a) to (e) respectively are: 80.4%, 81.4%, 88.4%, 82.2% and 86.2%.

and α produce best success rates because it balances learning and randomness during training. Additionally, moving slightly faster from randomness (i.e. exploration) to exploitation provides better success rate.

Future work will focus on building a general agent that is capable of exploiting many vulnerabilities and making the appropriate decision, which would expand and extend the capability of training an agent with specific and limited tasks. Our goal is to enhance our agent by incorporating Common Weakness Enumeration (CWE), which categorizes vulnerabilities by type, into our Q-Table. By having this capability, our agent can focus on vulnerabilities that are alike, rather than just on comparable operating systems. Also, the general agent will be equipped an arsenal of actions besides payloads that allows it to identify and pass through firewalls and anti-malicious solutions. Additionally, we expect the implementation of the general agent to grow in complexity and hence replace the Q-table with Deep Q-Network (DQN).

References

[1] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3826–3839, 2020.

[2] R. S. Sutton and A. G. Barto, "Reinforcement learning: an introduction mit press," *Cambridge, MA*, vol. 22447, 1998.

[3] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[4] G. Weidman, *Penetration testing: a hands-on introduction to hacking*. No starch press, 2014.

[5] M. Bishop, "About penetration testing," *IEEE Security & Privacy*, vol. 5, no. 6, pp. 84–87, 2007.

[6] Y. Stefinko, A. Piskozub, and R. Banakh, "Manual and automated penetration testing. benefits and drawbacks. modern tendency," in *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*, 2016, pp. 488–491.

[7] D. Kennedy, J. O'gorman, D. Kearns, and M. Aharoni, *Metasploit: the penetration tester's guide*. No Starch Press, 2011.

[8] "Metasploit," 2023, url: https://www.metasploit.com/, last accessed: Febreuary 10, 2023.

[9] "Metasploit RPC API," 2023, url: https://docs.rapid7.com/metasploit/rpc-api/, last accessed: Febreuary 10, 2023.

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

[11] R. Maeda and M. Mimura, "Automating post-exploitation with deep reinforcement learning," *Computers & Security*, vol. 100, p. 102108, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820303813

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

[13] S. Niculae, D. Dichiu, K. Yang, and T. Bäck, "Automating penetration testing using reinforcement learning," 2020.

[14] Z. Hu, R. Beuran, and Y. Tan, "Automated penetration testing using deep reinforcement learning," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2020, pp. 2–10.

[15] J. Schwartz and H. Kurniawati, "Autonomous penetration testing using reinforcement learning," *arXiv preprint arXiv:1905.05965*, 2019.

[16] "Apache CouchDB," 2023, url: https://couchdb.apache.org/, last accessed: Febreuary 10, 2023.

[17] "CouchDB Vulnerability," 2023, url: https://www.rapid7.com/db/modules/exploit/linux/http/apache_couchdb_cmd_exec/, last accessed: Febreuary 10, 2023.