Catch Me if You Can: Improving Adversaries in Cyber-Security with Q-Learning Algorithms

Arti Bandhana[a], Ondřej Lukáš[b], Sebastian Garcia[c] and Tomáš Kroupa[d]
Czech Technical University, Czech Republic

Keywords: Q-Learning, Reinforcement Learning, MDP, Cybersecurity, Learning Agents, Advanced Persistent Threat.

Abstract: The ongoing rise in cyberattacks and the lack of skilled professionals in the cybersecurity domain to combat these attacks show the need for automated tools capable of detecting an attack with good performance. Attackers disguise their actions and launch attacks that consist of multiple actions, which are difficult to detect. Therefore, improving defensive tools requires their calibration against a well-trained attacker. In this work, we propose a model of an attacking agent and environment and evaluate its performance using basic Q-Learning, Naive Q-learning, and DoubleQ-Learning, all of which are variants of Q-Learning. The attacking agent is trained with the goal of exfiltrating data whereby all the hosts in the network have a non-zero detection probability. Results show that the DoubleQ-Learning agent has the best overall performance rate by successfully achieving the goal in 70% of the interactions.

1 INTRODUCTION

The risk of cyber attacks is constantly increasing. Attackers continue to become more sophisticated and manage to find new vulnerabilities to exploit, making the role of network defenders skewed and asymmetric. Most attack techniques involve little direct interaction between the attacker and the defender. In attacks such as ransomware (ENISA, 2022), port scanning or cryptocurrency mining, the interaction can be as little as only one action from the attacker. In more complex attacks such as banking trojans or Advanced Persistent Threat (APT) attacks (Drašar et al., 2020), the attacker has to perform a series of steps within the network or target device to be successful while remaining undetected. Such attacks are extremely difficult to detect, yet they are the most impactful. APT attacks are usually long-term, with many decisions typically taken by a human adapting their tactics and techniques to avoid detection and in most cases, the defense mechanisms are not versatile enough to adapt to the behavior of an attacker.

APT attackers can be modeled as agents who pursue their goals while interacting with an environment (target device or network). Most of these interactions are captured mainly by Game theory or Reinforcement Learning (RL) models with the intent of improving defenses in the network. Game-theoretic frameworks are used to provide solutions for optimal defenses (such as honeypot allocation) but RL models are mostly used to improve penetration testing attacks (Durkota et al., 2016; Mitchell and Healy, 2018). LSTM network and Q-Learning techniques are also being applied to predict the attacker's action in APT data sets (Dehghan et al., 2022). However, modeling realistic defenses inevitably requires learning almost optimal decisions for attackers. To the best of our knowledge, there are no studies about modeling APT attacker's behavior with the goal to *improve* the decisions made by the attacker. Creating a realistic inference model for the attacker requires consideration of factors such as intent, capabilities, objectives, opportunities, and available resources for the attacker (Moskal et al., 2018; Liu et al., 2005). Due to the complexity of these attributes, developing a general framework becomes challenging. To overcome these challenges, RL models are generally applied to train and solve an optimal policy from a defender's perspective; however, we are unaware of a RL model to optimize the actions of an APT attacker.

In this paper, we model both an APT attacker and a network environment to train RL agents that optimize the attack. The goal of the attacker is to exfiltrate

[a] https://orcid.org/0000-0002-3711-3645
[b] https://orcid.org/0000-0002-7922-8301
[c] https://orcid.org/0000-0001-6238-9910
[d] https://orcid.org/0000-0003-1531-2990

Bandhana, A., Lukáš, O., Garcia, S. and Kroupa, T.
Catch Me if You Can: Improving Adversaries in Cyber-Security with Q-Learning Algorithms.
DOI: 10.5220/0011684500003393
In *Proceedings of the 15th International Conference on Agents and Artificial Intelligence (ICAART 2023) - Volume 3*, pages 442-449
ISBN: 978-989-758-623-1; ISSN: 2184-433X

data from a specific server inside a local network to a command and control (C&C) server in the Internet. To find the optimal policy for the attacker, three off-policy RL algorithms are trained: Q-Learning, Naive Q-Learning, and DoubleQ-Learning.

Our results show that the DoubleQ-Learning-based attacker agent is able to exfiltrate data in almost 70% of the interactions.

Furthermore, we show that the agent can learn how to plan and execute a multistage data exfiltration attack detected less than 40% of the time. From a cybersecurity point of view, it means that a model of an attacker can be learned and improved, and therefore a better model of the defender could be learned in future research.

The main contributions of this paper are:

- a novel model of a decision-making entity (APT attacker) in an adversarial environment;
- implementation of RL algorithms for an attacking agent in a custom environment; and
- an analysis of the impact of APT attacker models on the cybersecurity domain.

The paper is structured as follows. Section 2 provides the motivation and previous work. Section 3 describes the RL environment. Section 4 presents the RL algorithms; Section 5 presents the setup of the experiments; Section 6 presents the results and discusses their impact. The conclusions and future work are contained in Section 7.

2 MOTIVATION & RELATED WORK

There are two main sources of motivation for studying the behavioral models of attackers in APT attacks for local networks. First, improving defense mechanisms (algorithms, antivirus systems, etc.) based on the knowledge of past attacks highlights the need to better understand the characteristics of nearly optimal attack behaviors in realistic networks. Second, by creating and training RL models of the attacker's behavior, it is possible to optimize future defense mechanisms and the dynamic properties of such systems.

Game theory and RL (Shiva et al., 2010) have gained traction over the years in modeling attack and defense mechanisms in many domains, including network security.

Network security problems are primarily complex and require rational decision-making. Game theory provides mathematical models of strategic interaction among multiple decision makers (players or agents) along with algorithms for finding solutions (equilibria) in such scenarios. The potential benefit of applying game theory to network security is the automation of the exhaustive threat detection process for network administrators. However, real-world cybersecurity models may have limitations with regard to the information observed by players. Typically, the defender's knowledge of the attacker's strategy and decisions is limited (Patil et al., 2018). This leads to games with partial observation or incomplete information, which are extremely difficult to scale to the required size of the problem.

In the area of game theory for security, there has been promising research in honeypot technologies (Anwar and Kamhoua, 2022). The authors designed an optimal approach for honeypot allocation by formulating a two-player zero-sum game between the defender and the attacker, which is played on top of an attack graph. The defender places honeypots on machines, while the attacker selects an attack path through the attack graph, which would lead to the target machine without being detected. In addition to solving an effective strategy for honeypot placement in the network, the authors also experiment with a diversity of honeypot configurations. Diversifying the honeypot configuration ensures that not all honeypots are discovered if one is compromised; however, this adds to the operational cost. To automate response to a cyber attack, (Hammar and Stadler, 2020) investigate methods where strategies evolve without human intervention and do not require domain knowledge. The authors model the cyber interaction as a Markov game and use simulations of self-play where agents interact and update their strategies based on experience from previously played games.

Another promising research direction used Proximal Policy Optimization (PPO) with self-play to solve a stochastic (Markov) two-player game with sequential moves between defender and attacker (Du et al., 2022). The game is played on top of an attack graph, and the authors show that the performance of a PPO policy is better than that of a heuristic policy. The initial results are promising, but the setting used by the authors is limited to the attack graph with five nodes and four edges. By contrast, our work deals only with a single-agent environment.

Attack graphs are helpful, as they can predict the attacker's path depending on the vulnerabilities present in the network. At the same time, defenders can leverage attack graphs to find an effective defense strategy. In particular, (Guo et al., 2021) provides defense solutions through edge blocking in an attack graph constructed in the active directory. Another stream of research focuses on the assistance of

attacking tools for better penetration testing or cyber-training, for example, using Deep Q-Learning (Niculae et al., 2020). The authors compare Q-Learning, Extended Classifier Systems (XCS), and Deep Q-Networks (DQN) to find attacker strategies. To determine the best response for a suspicious user on the network, (Chung et al., 2016) compares the variations of Q-Learning with a stochastic game.

3 ENVIRONMENT MODEL

Q-Learning is one of the most widely applied model-free off-policy RL algorithms (Jang et al., 2019a). It allows agents to learn in domains with Markovian properties and thus can be modeled as a Markov Decision Process (MDP). Sufficient exploration of the environment is done with a ε-greedy policy. An ε-greedy method chooses a uniformly random action with probability ε and greedy action with probability $1-\varepsilon$. The hyperparameter ε is chosen to balance exploration and exploitation, intending to maximize the cumulative reward.

An MDP is used as the underlying model (Sutton and Barto, 2018) as the focus is on training a single attacking agent. Such an approach results in the defender being part of the environment. In real-life scenarios, successful detection requires several steps, from placement of the defensive measures, detecting and generating alerts, to evaluating and addressing threats. In this work, the defender is modeled as a stochastic and global part of the environment.

3.1 Network

The computer network used for the definition of the environment represents a small organization with five clients, five servers, and a router that provides Internet; see figure 1. Each host in the network has Internet access. The router is also a firewall that controls which clients from subnetwork 2 can access the servers in subnetwork 1 (corresponding to dotted lines in figure 1). Computers can connect to each other if they are in the same subnetwork.

In the environment, we assume that the attacker has already gained access to one of the clients on the network. Additionally, the attacker knows the address of an external C&C server on the Internet. The attacker's goal is to find and exfiltrate data located in one of the servers in subnetwork 1.

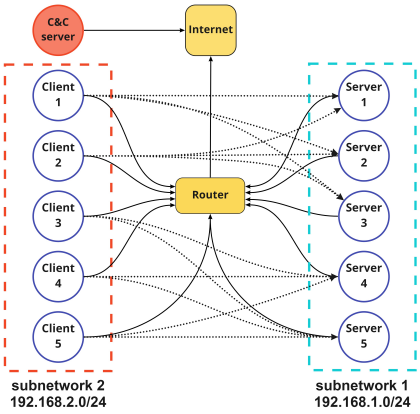

Figure 1: Network topology with two local subnetworks and a C&C server on the Internet. The solid black lines represent direct network connectivity (such as Ethernet cables). The dotted lines represent logical connections from clients to servers as allowed by the firewall. In the non-randomized experiments, the attacker starts in Client 1. In the experiments with a randomized start, the attacker starts in one of the clients in subnetwork 2. The IP address of the C&C server is always known to the attacker.

3.2 Defender

The defender in our model is an entity present in all clients/servers simultaneously and it has assigned a probability of detecting the attacker's action. Once the attacker is detected, the episode ends and the environment is reset to the initial state. This is represented by a terminal state in the environment. Given that the defender has full network visibility, there is a probability of detection for each action on all clients and servers.

3.3 Attacker

Attackers usually do not have information about the network and so they must compensate for lack of knowledge by learning through trial and error. We simulate an attacker who has already gained a foothold in subnetwork 1 (figure 1) according to our assumption. This holds for a real-world scenario, as the initial breach can be done in various ways since there are many connected devices on the network, and preventing the initial breach in some ways is extremely hard. Therefore, modeling the attacker entry in our current setup is ignored. The attacker's objective is to find the optimal path to a server in subnetwork 2 containing sensitive data, find and exfiltrate this data, and make it accessible on the web. The available actions are the minimal actions required to complete the goal: find hosts, find services, get access, find data, and exfiltrate. The attacker was modeled as a rational attacker behaving optimally.

3.4 States

A state is an abstract representation of the environment from the attacker's perspective. It contains several assets the attacker can use or has discovered with previous actions. Therefore, the state of the environment changes based on the actions of the attacker and the current state. The probabilities $p(a|s)$ represent the probability of success of the attacker's action a in a state s and $p(detection|s,a)$ represents the probability of detection given the action a played in the state s. These probabilities (table 1) of success and detection were set based on the expert evaluations of penetration test professionals, where knowledge of the domain was compared and matched with the evaluation of various detection tools for malicious behavior discovery shown in (Huang et al., 2022).

Table 1: Probability of success and detection for each action executed by the attacker in the network.

Action	Success probability	Detection probability
ScanNetwork	0.9	0.2
FindServices	0.9	0.3
ExecudeCode InService	0.7	0.4
FindData	0.8	0.1
ExfiltrateData	0.8	0.1

Success probabilities are based on known tools and techniques. While network issues are the cause of the failure of most actions, in the case of *ExecuteCodeInService* other problems such as service versions and exploits quality have to be taken into account. Detection probabilities consider the false positives found in real networks with benign traffic by a human player. Some actions, such as *ScanNetwork* with ARP scan, are highly successful and barely recognized (off-the-shelf state-of-the-art IDS can not detect it (Hou et al., 2010)). Often, even if these scans are detected, such alerts are dismissed for the sake of limiting the False positives. The same applies to *FindData* which is performed locally and thus nearly undetectable and *ExfiltrateData* which when done correctly, is known to be extremely hard to distinguish from benign traffic.
At each time step, the following information is part of the state:

- set of networks the attacker has discovered;
- set of hosts the attacker has discovered;
- set of hosts that the attacker has control of;
- set of services the attacker has discovered in each host; and
- set of data the attacker has discovered in a host.

Having states consisting of assets, we can follow the well-known STRIPS representation originally designed for planning (Fikes and Nilsson, 1971). STRIPS describes transitions in a system as operators, which are applicable if *preconditions* are met. Originally, the effects of *add* and *delete* can be specified for each operator. However, in our approach, we completely omit the delete effect, which results in a *relaxed* problem representation (Bonet and Geffner, 2001). Problem relaxation is a commonly used method in a variety of AI areas. Such an approach simplifies the problem of traversing the state space.

3.5 Actions

The attacker's actions follow the subset of techniques for adversary behavior listed in Mitre ATT&CK[1]. As we are only representing one type of goal in this model, data exfiltration, only the subset of Mitre actions related to data exfiltration are used:

1. active scanning:
   (a) find computers in the network
   (b) find services run on the hosts in the network
   (c) find data in the computer
2. attack service to execute code; and
3. exfiltrate data to the Internet.

The attacker in our model follows a five-step action as represented in Table 2 to reach its goal.

Table 2: List of actions and their effect on the network leading to a change in state.

Action	Description	Preconditions	Effects
ScanNetwork	Scans complete range of given network	network + mask	extends 'known_hosts'
FindServices	Scans given host for running services	host IP	extends 'known_services' with host:service pairs
ExecudeCode InService	Runs exploit in service to gain control of a host	host:service	extends 'controlled_hosts'
FindData	Runs code to discover data in a controlled host	hostIP	extends 'known_data' with host:data pairs
ExfiltrateData	Moves data from one controlled host to another	host:data:host	extends 'known_data' with 'target:data'

3.6 Rewards

The reward is an incentive that the agent receives with respect to the state action pair. In our model, the reward of the agent is constructed as: -1 for every action taken, -50 if the action is detected, and $+100$ if the goal state is reached.

The small negative reward per action is intended to motivate the agent to find the shortest path to the goal. The $+100$ reward for the achievement of the

[1]https://attack.mitre.org/

goal allows the attacker to take actions with a higher expected detection probability if they lead to a higher expected reward.

3.7 Implementation

The representation of a state, as described in section 3.4, allows the modification of the environment *without* the need to retrain the agent from zero. This differentiates our environment model and offers a higher degree of modularity for various cybersecurity scenarios. Instead of allocating the complete Q-table prior to training, our agents create the Q-values dynamically, saving both memory and time during training.

4 LEARNING AGENTS

To train and evaluate the attacker's performance, we use Q-Learning (Jang et al., 2019a) and its variants: Naive Q-Learning and Double Q-Learning. Q-Learning is a reinforcement learning algorithm that approximates the optimal state-action value function independently of the policy being followed. It is an off-policy algorithm that separates learning from the current acting policy by updating the Q-value $Q(s,a)$, which is an indication of how good a state-action pair is. The equation for the Q-value update is:

$$Q(s,a) \coloneqq Q(s,a) + \alpha(R_{t+1} + \gamma V^t(s')), \quad (1)$$

where $\alpha \in [0,1]$ is the learning rate and $\gamma \in [0,1]$ is the discount factor that captures the concept of depreciation. A value closer to 0 means that the current reward is preferred over future rewards.

In Naive Q-Learning, the learning rate is partially allocated to the previous result to combine the knowledge of the past history during learning, the actual immediate reward in the current iteration, and the expected future reward (Chung et al., 2016). This leads to the following variation of equation (1):

$$Q(s,a) \coloneqq \alpha Q(s,a) + (1-\alpha)(R_{t+1} + \gamma V^t(s')) \quad (2)$$

Double Q-Learning (Hasselt, 2010; Jang et al., 2019b) proposes learning two Q-functions instead of one. Each Q-function gets the update from the other for the next state. These two Q-functions are an unbiased estimate of the value of the action. The action selection is then performed by averaging or adding the two Q values for each action and then performing ε-greedy action selection with the resulting Q values. In this paper, action selection is performed by adding the two Q values before performing the ε-greedy.

$$Q^A(s,a) \coloneqq Q^A(s,a) + \alpha(R + \gamma Q^B(s',a') - Q^A(s,a)) \quad (3)$$

$$Q^B(s,a) \coloneqq Q^B(s,a) + \alpha(R + \gamma Q^A(s',a') - Q^B(s,a)) \quad (4)$$

The other two learning agents also use ε-greedy as the action selection criteria in accordance with the original papers.

5 EXPERIMENT SETUP

Three different scenarios were used to train the learning agents: specific attacker position, random attacker position, and random target server to attack.

In the first scenario, the attacker is placed on client 1 in subnetwork 2 (figure 1). We define a client as an official device on the network used for work and a server as a device that holds data and offers services accessed by the clients. The attacker's goal is to reach the target server, which is specified as server 3 in subnetwork 1; exfiltrate the data from the target server to the C&C server outside the local network.

There are five clients in subnetwork 2, and in reality, any connected device within the network is susceptible to an attack; therefore, for the second scenario, we randomly assign the starting position of the attacker. This was done to compare the performance of the learning agents and see how they adapt to randomness in the starting position. In addition to randomizing the starting position, we also randomized the target server for data exfiltration; which was our third scenario.

For successful achievement of the goal state, at least 5 successful actions had to be performed in all 3 scenarios; however, if the agent exceeds the limit of 25 actions per episode, the interaction is terminated.

The defender in all 3 scenarios is an entity with unlimited visibility and is present in all hosts, that is, every action can be detected with a predefined probability. Additionally, we assume that all services running on the hosts are exploitable and that a connection to the Internet is available on all hosts.

The learning parameter for each algorithm is presented in Table 3. Experiments start with a random attacker which randomly picks an action. The Q-Learning agent and the DoubleQ-Learning agent were trained on a learning rate of 0.3, while the Naive Q-Learning agent was trained on a learning rate of 0.8. The action selection parameter controlled by epsilon

was kept at 0.2 for all the agents; however, Double Q-Learning used a linearly decaying ε from 0.2 to 0.05.

In all experiments, we measured the win rate, the detection rate, and the mean return of the episodes. The win rate represents the percentage of interactions that were successful for the attacker, which is the number of times the attacker was able to reach the goal state and exfilter the data in 10 000 episodes. The detection rate represents the percentage of interactions that were detected and resulted in the attacker receiving a reward of −50.

Table 3: Training parameters: Q-Learning and DoubleQ-Learning agents were trained in 10 000 episodes, while NaiveQ-Learning was trained in 5 000 episodes. The discount factor γ was kept at 0.9 for all learning agents.

Algorithm	α	ε	γ	No. of episodes
Random	-	-	-	-
Q-Learning	0.3	0.2	0.9	10 000
Naive Q-Learning	0.8	0.2	0.9	5 000
Double Q-Learning	0.3	0.2	0.9	10 000

6 EXPERIMENTAL RESULTS

The following results were obtained in the first scenario of the experiment, when the attacker's position was specified in the network. Table 4 summarizes the performance of the different learning agents. The random attacker, without any knowledge of the network and without any strategy, has a detection rate of 99.58%, while the DoubleQ-Learning attacker had a detection rate of 33%. The Q-learning and Naive Q-Learning agents have similar detection rates.

Table 4: Performance comparison of the learning agents with a fixed attacker starting in *client1*. Q-Learning and Double Q-Learning were trained with 10 000 episodes, while Naive Q-Learning was trained with 5 000 episodes.

Algorithm	Winning rate (%)	Detection rate (%)	Mean return
Random	0.48	99.58	53.03
Q-Learning	66.4	40.4	43.94
Naive Q-Learning	66.91	40.19	43.94
Double Q-Learning	74.0	33.0	54.61

Randomizing the starting position decreases the win rate and increases the detection rate for all learning agents, as shown in table 5. The Naive Q-Learning agent had the greatest impact on performance due to the randomness of the starting position among all learning agents. The detection rate increased from 40.4% to 50.78%

Table 5: Comparison of performance for the learning agents in a scenario with randomized attacker's starting randomized. Q-Learning and Double Q-Learning were trained with 10 000 episodes, while Naive Q-Learning was trained with 5 000 episodes.

Algorithm	Winning rate (%)	Detection rate (%)	Mean return
Random	0.34	99.48	-54.04
Q-Learning	65.4	39.27	41.97
Naive Q-Learning	54.27	50.78	25.59
Double Q-Learning	68.9	36.8	47.58

Table 6: Comparison of learning agents in a scenario where the attacker's starting point and target server were randomized. All algorithms were trained on 10 000 episodes.

Algorithm	Winning rate (%)	Detection rate (%)	Mean return
Q-Learning	53.3	53	23.45
Naive Q-Learning	61.8	44.1	36.8
Double Q-Learning	64.9	41.7	41.2

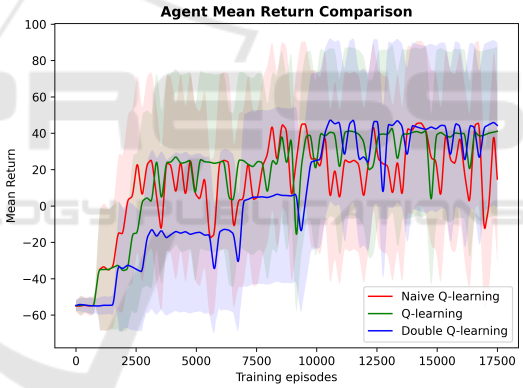

Figure 2: Comparison of the mean cumulative reward of agents during the learning process in the scenario with defender and randomized starting position for the attacker.

6.1 Analysis of Results

We compared how agents with varying parameters learned a policy in a network with ten hosts in the presence of a defender with full visibility. The detection probability was nonzero for actions at all clients and servers. When comparing the win rate and the detection rate for all learning agents, it is clear that Double Q-Learning outperforms all other agents in all scenarios. Two Q-functions are trained in such agent but from different episodes which makes the training more robust. A sum of the Q-functions is used during inference. This avoids the overestimation

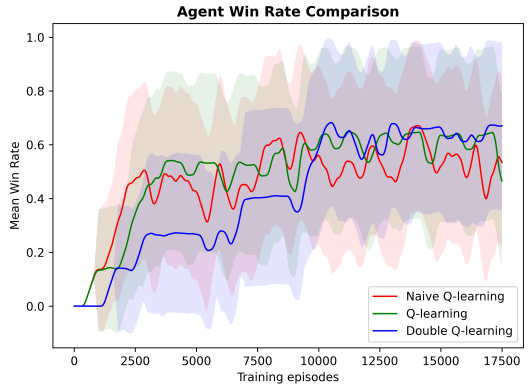

Figure 3: Comparison of the winning rate of agents during the learning process in the scenario with defender and randomized starting position.

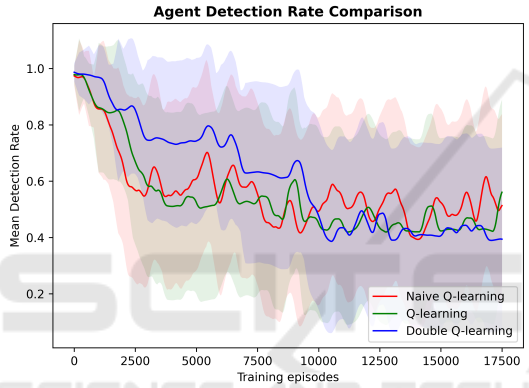

Figure 4: Comparison of the detection rate of agents during the learning process in the scenario with defender and randomized starting position.

bias of Q-Learning and leads to better training stability even in a noisy environment. The Q-Learning attacker and the Naive Q-Learning attacker have the same performance for the first scenario where the starting point was specified. This is due to the distribution of the learning rate according to equations (1) and (2). A learning rate of 0.8 was used for Naive Q-Learning, which in comparison with the Q-learning gives similar results as the learning rate of 0.2. However, the performance of Naive Q-Learning decreased when the starting position was randomized. This is attributed to the weighting of the update rule of the Q-value, as shown in equation (2). When considering negative rewards, the update affects the Q-value more than the standard Q-Learning due to the split update α. Although this can be beneficial in the cases of high positive rewards, the results show that this approach lacks adaptability in the case of the stochastic environment.

Figures 3 and 4 show that DoubleQ-Learning outperforms the other two agents in terms of winning and detection rates. The high variance of the mean returns, as shown in figure 2 is the result of the stochastic environment and the reward distribution described in section 3.6. The graphs also show that even though DoubleQ-Learning performs badly in the beginning, over time as the number of episodes increases and state-action values are updated, it outperforms the other two learning agents. In particular, even if the agent's policy is optimal, it cannot influence the detection and subsequent reward of -50. Therefore, the three agents share similar high variance in mean returns but differ significantly in metrics that focus on reaching the goal, in which the Double Q-Learning shows the most promising results.

Despite using random exploration ε in the three agents based on Q-Learning, the results from the first and second scenarios show that the environment and the goal are non-trivial and unsolvable for agents performing purely random actions, which reached the goal in fewer than 1% of the cases. For that reason, the Random Agent was excluded from the comparison in figures 2, 3 and 4 and in the *third scenario*.

The results of our experiments show that despite the defender having full visibility of the network, a rational attacker was still able to reach the target and exfiltrate data. From a security perspective, this indicates that the defensive tools in the network need to be improved so as to prevent the attacker's lateral movement in the system.

7 CONCLUSION

In this paper, we propose a Q-Learning-based attacking agent capable of performing data exfiltration.

Our results show that even though the three learning agents can find meaningful policies, Double Q-Learning outperforms the others and provides the most stable training. It reached the goal 70% of the interactions while being undetected in 37%. This shows that despite a globally present defender, a rational attacker could still reach the target.

The initial success and detection probabilities were set based on expert knowledge, however, our results clearly show that there is room for improvement in the detection capability of the defender. Having a high success probability for attacker action highlights the need for a robust defense mechanism that is capable of detecting any stealthy attacker. This provides a foundation for studying and improving attacker techniques to increase defense capability in the network.

Currently, the method is limited to small or medium-sized networks. Although the interaction

and world representation model can be easily extended to a more complex setup in size of the network and action space, the scalability and computational feasibility of such extensions have yet to be evaluated.

Therefore, the natural direction for future research is to expand our approach towards larger environments, which will require subsequent scalability testing due to those complex setups. We also plan to incorporate other types of cyber attacks into Mitre taxonomy and model the defender as a rational entity with its own set of actions in the interaction. In addition, we plan to test the performance of our agent in a simulated environment.

Along with increasing the environmental complexity, the problem of more complex goals for the attacker is also in the pipeline resulting in the need for more reconnaissance from the agent.

ACKNOWLEDGMENTS

The authors acknowledge support from the Research Center for Informatics (CZ.02.1.01/0.0/0.0/16_019/0000765) and Strategic Support for the Development of Security Research in the Czech Republic 2019–2025 (IMPAKT 1) program, by the Ministry of the Interior of the Czech Republic under No. VJ02010020 – AI-Dojo: Multi-agent testbed for the research and testing of AI-driven cyber security technologies.

REFERENCES

Anwar, A. H. and Kamhoua, C. A. (2022). Cyber deception using honeypot allocation and diversity: A game theoretic approach. In *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*.

Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1):5–33.

Chung, K., Kamhoua, C. A., Kwiat, K. A., Kalbarczyk, Z. T., and Iyer, R. K. (2016). Game theory with learning for cyber security monitoring. In *2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*, pages 1–8.

Dehghan, M., Sadeghiyan, B., Khosravian, E., Moghaddam, A. S., and Nooshi, F. (2022). ProAPT: Projection of APT Threats with Deep Reinforcement Learning. arXiv:2209.07215 [cs].

Drašar, M., Moskal, S., Yang, S., and Zat'ko, P. (2020). Session-level adversary intent-driven cyberattack simulator. In *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 1–9.

Du, Y., Song, Z., Milani, S., Gonzales, C., and Fang, F. (2022). Learning to play an adaptive cyber deception game. In *The 13th Workshop on Optimization and Learning in Multiagent Systems, AAMAS*.

Durkota, K., Lisy, V., Kiekintveld, C., Bosansky, B., and Pechoucek, M. (2016). Case studies of network defense with attack graph games. *IEEE Intelligent Systems*.

ENISA (2022). ENISA threat landscape for ransomware attacks. Technical report, ENISA, LU.

Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208.

Guo, M., Li, J., Neumann, A., Neumann, F., and Nguyen, H. (2021). Practical fixed-parameter algorithms for defending active directory style attack graphs.

Hammar, K. and Stadler, R. (2020). Finding effective security strategies through reinforcement learning and self-play. In *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE.

Hasselt, H. (2010). Double q-learning. *Advances in neural information processing systems*, 23.

Hou, X., Jiang, Z., and Tian, X. (2010). The detection and prevention for arp spoofing based on snort. In *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, volume 5, pages V5–137–V5–139.

Huang, Y.-T., Lin, C. Y., Guo, Y.-R., Lo, K.-C., Sun, Y. S., and Chen, M. C. (2022). Open source intelligence for malicious behavior discovery and interpretation. *IEEE Transactions on Dependable and Secure Computing*.

Jang, B., Kim, M., Harerimana, G., and Kim, J. W. (2019a). Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*.

Jang, B., Kim, M., Harerimana, G., and Kim, J. W. (2019b). Q-learning algorithms: A comprehensive classification and applications. *IEEE access*.

Liu, P., Zang, W., and Yu, M. (2005). Incentive-based modeling and inference of attacker intent, objectives, and strategies. *ACM Transactions on Information and System Security (TISSEC)*, 8(1):78–118.

Mitchell, R. and Healy, B. (2018). A game theoretic model of computer network exploitation campaigns. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*.

Moskal, S., Yang, S. J., and Kuhl, M. E. (2018). Cyber threat assessment via attack scenario simulation using an integrated adversary and network modeling approach. *Journal of Defense Modeling and Simulation*.

Niculae, S., Dichiu, D., Yang, K., and Bäck, T. (2020). Automating penetration testing using reinforcement learning.

Patil, A., Bharath, S., and Annigeri, N. (2018). Applications of game theory for cyber security system: A survey. *International Journal of Applied Engineering Research*, 13(17):12987–12990.

Shiva, S., Roy, S., and Dasgupta, D. (2010). Game theory for cyber security. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, pages 1–4.

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