Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Automating post-exploitation with deep reinforcement learning

*Ryusei Maeda**, *Mamoru Mimura*

1-10-20 Hashirimizu, Yokosuka, Kanagawa, Japan

ARTICLE INFO

Article history:
Received 19 March 2020
Revised 7 August 2020
Accepted 5 November 2020
Available online 10 November 2020

Keywords:
Reinforcement learning
Post-exploitation
A2C
Q-Learning
SARSA
Deep reinforcement learning
Lateral movement

ABSTRACT

In order to assess the risk of information systems, it is important to investigate the behavior of the attacker after successful exploitation (post-exploitation). However, the audit requires the experts, and to the best of our knowledge, there are no solutions to automate this process. This paper proposes a method of automating post-exploitation by combining deep reinforcement learning and the PowerShell Empire, which is famous as a post-exploitation framework. Our reinforcement learning agents select one of the PowerShell Empire modules as an action. The state of the agents is defined by 10 parameters such as type of account that was compromised by the agents. In the learning phase, we compared the learning progress of the 3 reinforcement learning models: A2C, Q-Learning, and SARSA. The result shows that the A2C could gain reward most efficiently. Moreover, the behavior of the trained agents are evaluated in a test domain network. The results show that the trained agent using A2C could obtain the administrative privileges to the domain controller.

1. Introduction

Much attention has been directed to information security with the development of information and communication technology. There are two types of security evaluation methods for network systems: penetration tests and red team tests. The penetration test performs a systematic vulnerability scan of the network, applications, hardware, etc. These types of tests create a matrix of vulnerabilities, patching issues, and very actionable results. Therefore, the penetration test is an effective method for evaluating the existence of known vulnerabilities. However, the real attacker does not perform vulnerability scanning as performed in the penetration test in the behavior after successful exploitation (post-exploitation). In the red team test, the red team acting as an attacker performs from intrusion to the post-exploitation according to the goal of the campaign. The target of evaluation is not limited to the vulnerability, but also the skill and security policy of the defender blue team. The post-exploitation consists of the following actions: lateral movement, privilege escalation, collecting information, and building backdoors, etc. These processes are performed as stealthy as possible to mimic the behavior of a real attacker. Altogether, the red team test can provide a more comprehensive security assessment than the penetration test. There are many solutions and vulnerability management programs that support and automate penetration testing, e.g., OpenVAS[1], sqlmap[2] and DeepExploit (Isao). On the other hand, to the best of our knowledge, there are no solutions to automate the post-exploitation of the red team test.

[1] https://www.openvas.org/
[2] http://sqlmap.org/

* Corresponding author.
E-mail addresses: cgrddntfr5@yahoo.co.jp (R. Maeda), mim@nda.ac.jp (M. Mimura).
https://doi.org/10.1016/j.cose.2020.102108

Incidentally, one of the well-known automation approaches for security solutions is the approach that uses machine learning. There are three types of machine learning: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning and unsupervised learning are used for intrusion detection, malware detection, privacy protection systems, etc (Apruzzese et al., 2018; Çavusoglu, 2019; Cui et al., 2018; Jia and Gong, 2018; Kim et al., 2019; Milosevic et al., 2017). These methods use datasets for training, e.g., malware dataset, review dataset and mail dataset. Preparing a large dataset for training is a prerequisite for the automation of security solutions. However, it is difficult to prepare the dataset of behavior in a real-time, continuous environment such as the post-exploitation. Therefore, supervised learning and unsupervised learning are not proper for the automation of the post-exploitation. Reinforcement learning is the machine learning type that learns with the exploration of environments and the accumulation of experience. The trained reinforcement learning agents are modeled, so as to perform continuous optimal actions. Thereby, the agents can be applied to complex and real-time environments. The previous works have demonstrated that multi-agent reinforcement learning can be applied to cyber-security simulation scenarios (Bland et al., 2020; Elderman et al., 2017; He et al., 2016). Besides, they proposed an algorithm to learn the optimal strategy. However, to the best of our knowledge, no study has been applied reinforcement learning to actual cyber-security scenarios. It is necessary to make the training environment (the simulation of the previous works) concrete and practical so that apply theirs works to a actual cyber-security scenario. Whereas, it is not realistic to prepare many concrete learning environments in the real world.

In this study, we apply the concept of data augmentation (Chawla et al., 2002; Leen et al., 2001; Simard et al., 2000), so as to cope with the lack of learning data and the environment. This can add diversity to the learning environment. Besides, it prevents overfitting of the reinforcement learning agents. The trained reinforcement learning agents are designed to automate efficient post-exploitation in a real environment. The contributions of this paper are as follows.

- Evaluating the applicability and effectiveness of deep reinforcement learning to the post-exploitation in a real environment
- Automating red team testing tasks

This paper is organized as follows. Section 2 provides an overview of security techniques using reinforcement learning. Section 3 provides an overview of the reinforcement learning model used in our work. Section 4 presents the details of our method. Section 5 explains the experimental setup and the results. Section 6 discusses these results. Finally, in Section 7 the main findings are summarized.

2. Related work

2.1. *Lateral movement*

This paper focuses on the automation of lateral movement among the actions that constitute the post-exploitation (Section 4.4, Section 5). Once APT (Advanced Persistent Threats) attackers entered the target networks, they cautiously use the compromised systems as stepping stones so that reach critical systems buried deep inside the networks. These incremental movements to critical systems in an inside network are called lateral movement. Many network-based and host-based solutions (such as Windows Defender, McAfee, Norton, Snort, and OSSEC) are developed to detect and eliminate lateral movement. Besides, there have been considerable studies on this theme. Tian et al. proposed the method that efficiently detects the lateral movement in a complex edge cloud computing environment (Tian et al., 2019). Lah et al. proposed the framework for improving the detection of the lateral movement based on pattern risk scoring (Lah et al., 2018). On the other hand, the lateral movement methods of attackers are diversifying. The popular methods are as follows: exploiting vulnerabilities in SMB and RDP services, exploiting credentials (e.g. credential dumping, pass-tha-hash, pass-the-ticket (B.Deply (2014),Duckwall and Campbell,Dunagan et al. (2009))), reusing existing client communication (e.g. SSH hijaking (Boileau)). Niakanlahiji et al. have proposed a stealthy lateral movement method that does not require privilege escalation or establishing a new connection (Shadowmove, 2020). The lateral movement requires high skill because it needs additional operation such as credential dumping. The purpose of this study is to automate the real lateral movement by deep reinforcement learning. Simple automation of the lateral movement (e.g. brute force attempts with scripts) does not correspond to actual attacker actions. Therefore, this paper proposes an efficient and more realistic automation method. In our method, the reinforcement learning agent learns the most suitable strategy and technique for the defending system.

2.2. *Applying reinforcement learning to penetration tests*

Ghanem and Chen (2018) proposed an intelligent penetration testing approach using reinforcement learning. The proposed system is modelled as a partially observed Markov decision process (POMDP), and tested using an external POMDP-solver. The results support the hypothesis that reinforcement learning can enhance penetration testing in term of accurate and reliable outputs. However, the work of Ghanem et al. is limited to only the planning phase and not entire implementation phase in actual environment.

DeepExploit (Isao) is the fully automated penetration test framework linked with Metasploit (Rapid7). DeepExploit identifies the status of all open ports on the target server and executes the exploit at pinpoint using reinforcement learning in actual environment.

The goals of these frameworks are only automating and improving vulnerability diagnoses and initial exploitation tasks, these frameworks do not support the post-exploitation. On the other hand, the goal of our work is automating post-exploitation and improving the efficiency. In the automation of the initial exploitation task, no state transition occurs for the reinforcement learning agent. By contrast, in the automation of the post-exploitation task, the state transition of the agent must be defined. Specifically, the post-exploitation task is represented as a sequential process, therefore, the state

transition occurs. Our method applies reinforcement learning to the sequential process.

2.3. Cyber security simulation

Elderman et al. (2017) focus on a cyber-security simulation game in networks. The game is an adversarial sequential decision making problem played with two agents, the attacker and defender. The simulation network configuration is modeled as the network composed of nodes where the attacker and defender move. The state of the attacker is the node where the attacker is located. Each of attacker's actions has an attack value, and the exploit succeeds when it is larger than the defense value of the defender. The two agents pitted one reinforcement learning technique against each other and examined their effectiveness against learning opponents. This work showed that the agents are not able to win over the long term because of both agents trying to adapt each other.

However, there are actually few situations like the previous work: in the situations, the attacker and defender adapt and deal with each other in real-time. According to Sharma et al. (2011), 62% of cyber attacks are detected after the attackers achieved their goals. This means that attackers and defenders rarely compete in real time. Therefore, in this paper, we train the attack agents in the environments that does not explicitly set the defense agents.

Bland et al., 2020 implemented a reinforcement learning algorithm to the cyberattack models that were modeled using an extension of the Petri net formalism. The models were validated by a panel of cybersecurity experts in a structured face validation process. Therefore, this simulation is more realistic than the cyber-security simulation game. The experiments were conducted with an attacker and defender competing each other. The results demonstrated the potential of formally modelling cyberattacks and of applying reinforcement learning to improving cybersecurity.

Both of the previous works have been validated the effects of reinforcement learning only by the simulation. The second contribution of our work is to validate the effects of reinforcement learning in a real environment, not a simulation. Our work embodies the previous works in the following points: the network configuration, the agent state, and the agent actions. This allows the trained agents to work in actual environment.

3. Reinforcement learning

Our method uses deep reinforcement learning to automate the post-exploitation. This section provides an overview of reinforcement learning and A2C, the algorithm used in our method.

3.1. Types of reinforcement learning

Reinforcement learning algorithms are designed to efficiently explore the optimal policy under a given environment. The value-based algorithms focus on estimating the optimal action value function. The agents of the value-based algorithm use the experience gained from the environment to update the value evaluation. These algorithms are represented by Q-Learning. In contrast, the policy-based algorithms focus on improving current policies. The agents of the policy-based algorithm use the experience to update their strategies. These algorithms are represented by State-Action-Reward-State-Action(SARSA). In addiion to these 2 types, there is the Actor-Critic that combines a value-based method and a policy-based method. The Actor-Critic method is based on the idea that the action values and strategies can be considered separately. Thus, This method updates the strategy and value evaluation separately to advance learning. This method generally takes more time for learning than the other methods. However, in the end, this method has a more stable learning result than other methods.

3.2. A2C

Our method uses A2C (Advantage Actor Critic) as a reinforcement learning. A2C is similar to A3C (Asynchronous Advantage Actor Critic) (Mnih et al., 2016), but there is no asynchronous part. Both A2C and A3C are learning algorithms using the advantage. The advantage is expressed in the following equation: $A(s, a) = Q(s, a) - V(s)$. The advantage is denoted as $A(s, a)$. We denote the state of the agent as s and the action as a. $V(s)$ represents the pure value of the state s ; therefore, $A(s, a)$ represents the pure value of action a. Considering the advantage can stabilize the learning. According to OpenAI, A2C lacks the asynchronous part but A2C performs better than A3C (Wu et al.). Thus, our method uses A2C, so as to stabilizes and improves the learning progress.

In our experiments, SARSA and Q-Learning were implemented in addition to A2C so that demonstrate the superiority of A2C.

4. Proposed method

4.1. Overview

This section describes the proposed method, that is, the components of the training of the reinforcement learning agents. First, we define the state of the agent s. Second, we define the action a selected by the agent in the environment (selected from the action list A). The modules registered in PowerShell Empire are set as action list A. Lastly, we set rewards r according to the result of the action a. Fig. 1 shows an overview of our method. The agent accumulates the set of s, a, and r observed from the learning environment as experience and proceed with learning. Our method uses A2C as the model for reinforcement learning. We build multiple environments to make a distributed environment. The learning efficiency is improved by the distributed collection of experience.

This section is organized as follows. Section 4.2 describes the agent state s. Section 4.3 describes the action a. Section 4.4, describes the rewards r setting. Finally, we describe the implementation in Subsection 4.5.

4.2. Definition of agent state

The state s of the agent is defined by 10 entries. Table 1 shows the names of the entries and the summary of its information.

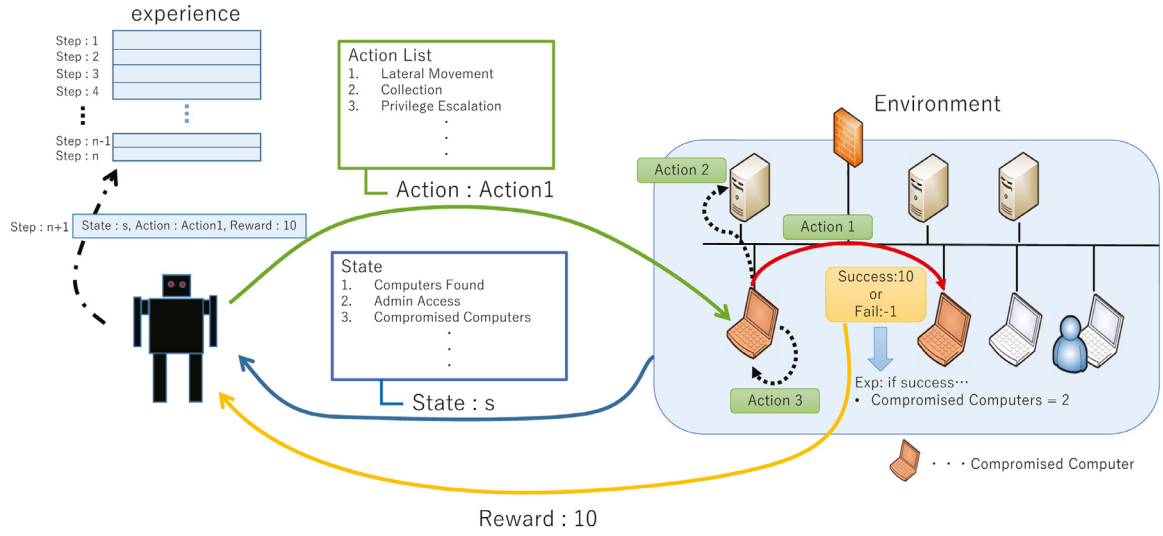

Fig. 1 – The overview of the proposed method. The components of the training environment are action, state, and reward. The state of the agent is defined in the environment, and the action is determined based on the experience. The agent gains rewards according to the result of the action. The agent determines the next action corresponding to the state based on the experience.

Each entry is the information that can be observed in both the actual environment and the training environment. The information of the entries represents information possessed by the real attackers. This makes it possible to reproduce the realistic attacks. Discovered Computer denotes the number of discovered computers in the network. Likewise, Compromised Computer denotes the number of compromised computers in the network. These indicate the current location of the agent in the network and the progress of the campaign. Besides, each module has a unique number. The value of the Previous Module entry is the number of the module performed in the previous step. The Admin Access value indicates whether any other computers can be accessed using compromised user privilege. If such computers are discovered, the agents can perform the lateral movement and code execution on the computers. Therefore, this information should be actively used. The entry of User Name, Password, and Hash represents the capture status of the credential information. These credentials are useful for the agents to compromise new computers and other assets in the network. The entry of Rhost indicates whether a vulnerable host was discovered in the network. DCOM (Distributed Component Object Model) is a Windows feature for communicating between software components on different remote computers. There are some methods for the lateral movement that the agents can take advantage of by using the DCOM. If other methods of the lateral movement are being monitored, the agents can take advantage of DCOM. Altogether, the presence of the DCOM application influences the determination of the agents. Hence, we add the DCOM entry as one of the elements of the agent state.

Table 1 – The definition of the state of the agent.

Entry Name	Details
Discovered Computer	Number of discovered computers in the environment
Local Admin Access	Whether the agent has local admin access or not (define with 0 or 1)
Compromised Computer	Number of compromised computers
Previous Module	The action performed in the previous step
User type	Type of the user: User or privileged user (define with 0 or 1)
User Name	Whether the agent capture user names in the environment or not (define with 0 or1)
Password	Whether the agent capture plaintext credential or not (define with 0 or 1)
Hash	Whether the agent capture hash of credential or not (define with 0 or 1)
Rhost	Found vulnerable hosts in the network or not (define with 0 or 1)
DCOM	Found running DCOM Applications or not (define with 0 or 1)

4.3. *Definition of agent action*

We define the actions that are actually performed in the post-exploitation as the agent actions, so as to automate practical tests. Recently, PowerShell is often used in the post-exploitation (Wueest). PowerShell Empire (Powershell empire) is a well-known tool implemented as the post-exploitation framework using PowerShell. For this reason, we set the PowerShell Empire modules (204 modules) as the agent action list A. A is classified into 12 groups by the characteristics. Table 2 shows the group names after the classification of A and the number of modules registered in the group. The modules belonging to the same group have different means and mechanisms. However, they are basically used to achieve the same purpose. For example, the performance of a module for capturing a screen differs from that of a module for installing a

Table 2 – PowerShell Empire Modules Classification.

Group name	Number of registered modules
Code Execution	6
Management	30
Collection	24
Persistence	18
Credentials	23
Privesc	22
Exfiltration	2
Recon	3
Exploitation	3
Situational Awareness	52
Lateral Movement	12
Trollsploit	9

keylogger; however, they belong to the same group because they are used for information gathering. Collection group is the group of modules for information gathering by the method such as the key logger described above. Credential group is the module group that can capture credentials and tokens using Mimikatz etc. Lateral Movement group is the group of modules that perform the lateral movement using various methods such as DCOM, WMI (Windows Management Instruction), PS command (PowerShell session command), etc. We briefly described the 3 categorized groups here. More information can be obtained on the PowerShell Empire project official page (Powershell empire).

4.4. *Reward setting*

This section describes the reward setting. We set the rewards for each module group separately. As we described in Section 4.3, the modules are classified according to the purpose. In the training environment, the rewards can be set according to the goals that the agents should achieve. In the testing phase of Section 5, the goal of the agents is to obtain the administrative privilege to the domain controller. In that case, the first compromised computer is usually not an asset like a domain controller. Consequently, in order to obtain the privilege or discover valuable systems, the attackers must move around the network. For this reason, the modules of the Lateral Movement group are the key actions in this campaign. The reward r to the agents is set to be given for the success of the lateral movement. Besides, the size of r depends on whether a new account control has obtained at the time of the lateral movement success. For example, even if the lateral movement between computers using WMI is successful, the varieties of the actions that can be performed in the network do not change. If the agent obtains control of an account with higher privilege, the probability of access to high-value assets increases, that is, the value of the success is high. Therefore, the reward for the success of the high-value lateral movement is set higher than the others. Reward r setting is as follows.

- r=50 if the valuable Lateral Movement is successful.
- r=10 if the low-value Lateral Movement is successful.
- r=-1 if the lateral movement fails.
- For other actions, r=-1 regardless of success or failure.

Reward r=-1 means punishment for the agents. The agents try to maximize the reward. Therefore, if r=-1 is set for each action, the agents try to reach the goal as soon as possible. This corresponds to a situation where real attackers achieve the goal as soon as possible to avoid detection of the breach. Besides, the episode ends when the agent obtains the reward r=50, that is, when the agent succeeds the high-value lateral movement.

4.5. *Implementation*

Fig. 2 shows an overview of the implementation. The hidden layer of a deep neural network is 3 fully connected layers. Thus, the deep neural network is consists of a total of 5 layers including the input and output layers. The input value is the agent state s. The output values are the selection probability $p(a)$ of each action included in the action list A and the state value $v(s)$. In other words, the output values are the probability distribution of the action selection, and the state value. Action options of the agents are modules acquired via RESTful API implemented in PowerShell Empire. Deep neural networks accumulate experience that is composed of a set of "the selected actions, the states before and after actions, and the rewards". Afterward, it computes the gradient between each node by using the experience. The action selections of the agents are based on the ε greedy method. According to the value of ε, a that $p(a)$ is max or randomly determined is executed. The value of ε is set to 0.5 at the start of the training and approach 0 in proportion to the progress of the training. We used the Restful API implemented in PowerShell Empire for communication between the agents and PowerShell Em-

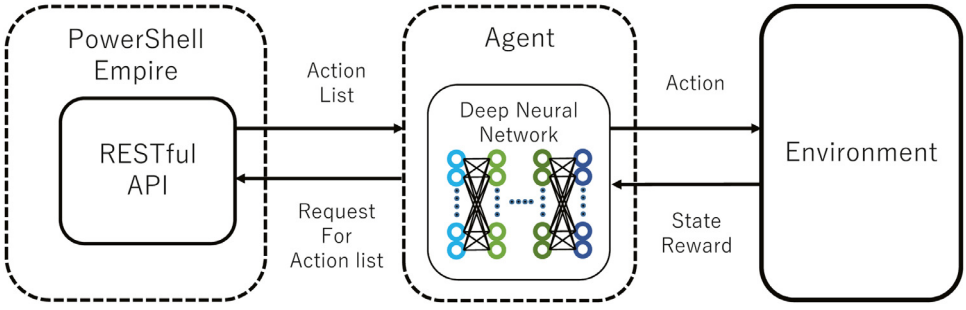

Fig. 2 – Implementation overview. Because A2C is implemented, there are actually multiple environments at the same time.

pire. Our method including the definition of the state, communication with RESTful API, and the setting of rewards, was implemented using Python 3.6.

5. Evaluation

5.1. *The learning phase*

5.1.1. *Setting up training environment and training the agent*

Setting up a large amount of environment for the training of the agents is not practical. If a large-scale environment could be prepared in the real world for the agent training, learning from the real world sample would have been possible: like Deep Exploit. This situation is the ideal environment for this study; however, we could not solve this problem due to practicality and cost issues. Besides, there is no public resource for the training of the agents to the best of our knowledge. To address these issues, we designed the training environment so that the network configuration and the vulnerability were not uniquely determined. Specifically, we applied the concept of data augmentation (Chawla et al., 2002; Leen et al., 2001; Simard et al., 2000) and added noise to the basic network settings, so as to give diversity to the environment.

In this study, the goal of the learning phase is to learn the task of obtaining the administrative privilege to the domain controller. There are several tasks to learn. In this paper, we chose this task for the sake of simplicity. The key actions to achieve the goal are the lateral movement as described in Section 4.4. We set the probability of success for the key actions, the lateral movement, so that the network setting and the vulnerabilities are not uniquely determined. That is, this is the noise for preventing the agents from over-fitting. When an agent attempts a lateral movement, the success or failure depends on the state of the agent and the success probability of the action. The amount of the reward give the agents depends on the success or failure. The agents attempt the post-exploitation, get the rewards based on their success or failure, and accumulate them as their training samples. Summary, the agent attempts the post-exploitation for each pattern of the probability settings, collects the training samples, and learns. We attempted 5 patterns of the probability setting so as to set the success probability that can perform appropriate learning: A (all success probabilities are set to 20%), B (the pattern of 50%), C (the pattern of 80%), D (the success probabilities are set randomly), and E (the success probabilities are set to a certain range according to the characteristics of the module). Table 3 shows the probability settings for E. Fig. 3 shows the transition of the obtained rewards of each method. The reinforcement learning model is A2C in any method. The horizontal axis represents the number of times the agent executed the module(action). We attempted 50,000 steps in each method. The results indicate A, B, and C were successful in the learning phase.

Table 3 – Success probabilities of modules performing Lateral Movement in E.

Module Name	Success Probability
invoke wmi debugger	40–60%
invoke wmi	70–90%
invoke sshcommand	40–60%
invoke psexec	40–60%
invoke psremoting	10–30%
invoke smbexec	10–30%
jenkins script console	10–30%
invoke dcom	70–90%
invoke executemsbuild	40–60%
inveigh relay	40–60%
new gpo immediate task	10–30%
invoke sqloscmd	10–30%

Fig. 4, 5, 6, 7, and 8 show the transition of the loss value in the learning phase of each method. The values of loss for B, C, and D show a lot of fluctuations and do not tend to converge to 0 even before the end of the learning. On the other hand, the loss values for A and E grow rapidly from when the obtained rewards grow. Besides, the loss values for A and E tend to converge to 0 as the obtained rewards level off. These re-

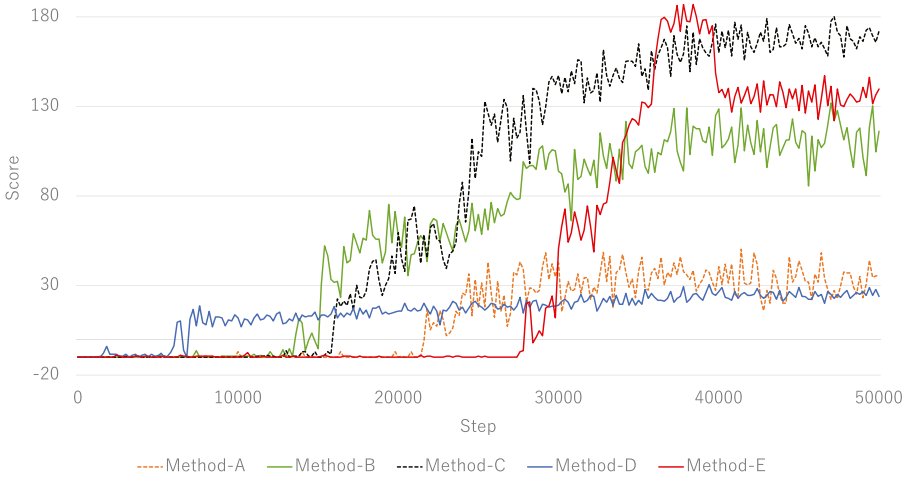

Fig. 3 – Comparison of the progress of learning. Each line in the graph indicates the methods of setting the success probability. We calculated the average reward per 10 steps from the reward of the last 200 steps.

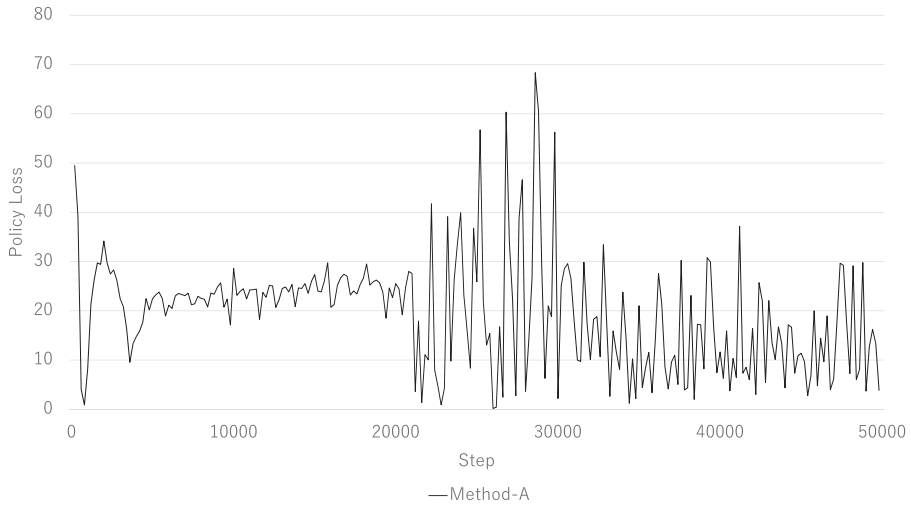

Fig. 4 – Transition of the loss values for A.

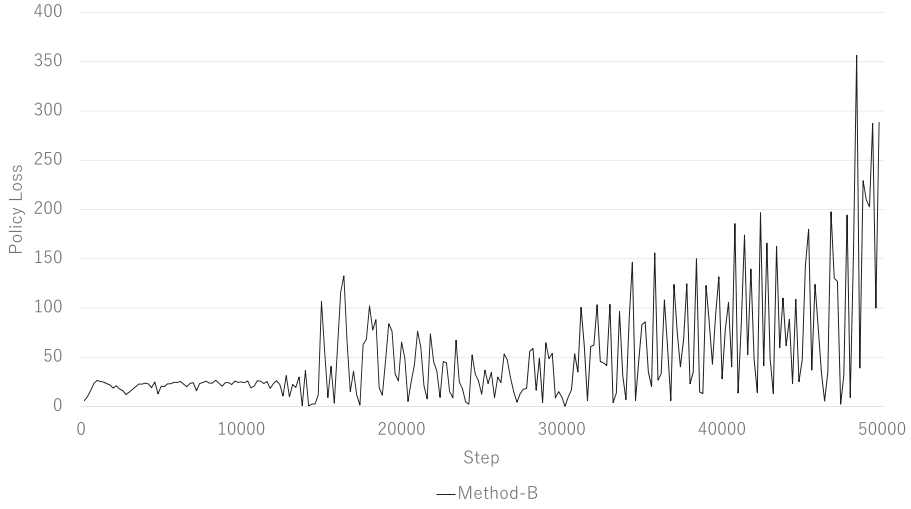

Fig. 5 – Transition of the loss values for B.

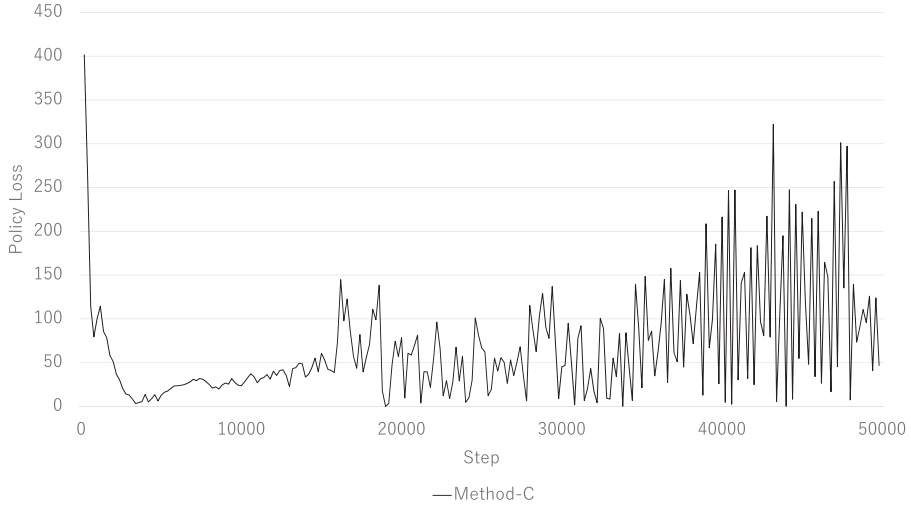

Fig. 6 – Transition of the loss values for C.

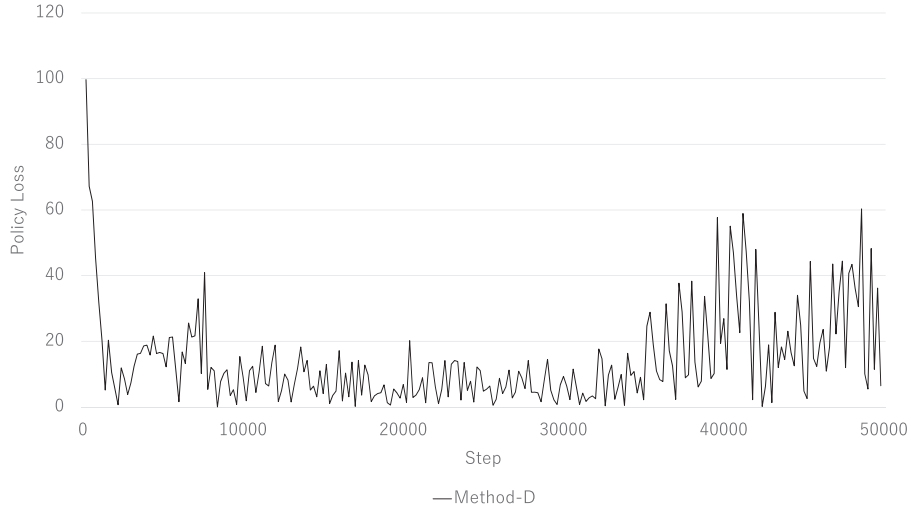

Fig. 7 – Transition of the loss values for D.

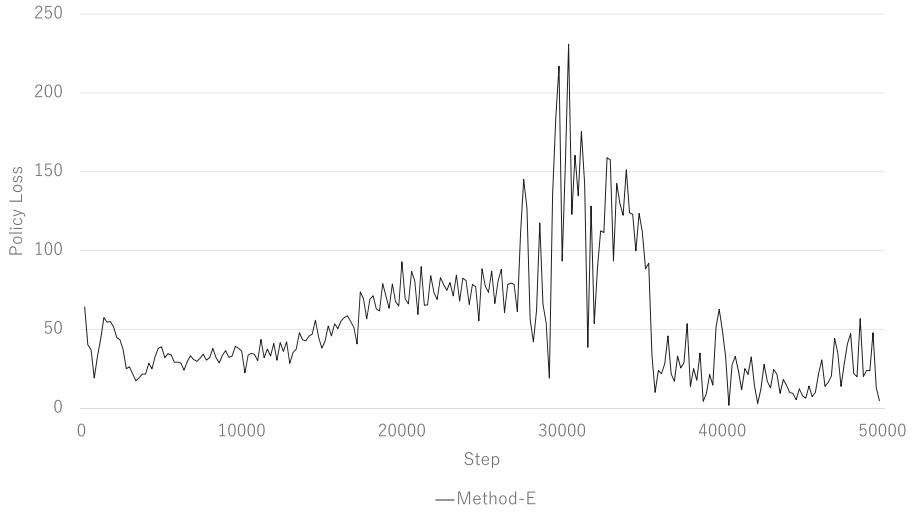

Fig. 8 – Transition of the loss values for E.

sults indicate that Method-A and Method-E are more suitable for training than Method-B, C and D in terms of the loss value.

5.1.2. *Comparison of reinforcement learning models*

Next, we evaluated the progress of training for the 3 models of reinforcement learning. In addition to A2C, Q-Learning and SARSA were implemented. We attempted the training of the Q-Learning agent and SARSA agent under the same conditions as for A2C. We applied C and E, which performed well in the A2C learning phase, to the Q-Learning and SARSA agent learning. Fig. 9 and Fig. 10 respectively show the transition of the gained rewards for each model when C and E are applied. The results show that the A2C model has higher final rewards than the other models. It is evident that A2C had better learning efficiency than other models. The SARSA's rewards are partially higher than the Q-Learning's rewards but not stable. Besids, there was no significant change in the gained rewards of Q-learning agents.

5.2. *The testing phase*

In this section, we compare the trained agents by running them in the real environment. The trained agents are executed in the test Windows domain network. The goal of the trained agent is to obtain the administrative privilege to the domain controller. We measure the number of execution steps and the run time until obtaining the administrative privileges to the domain controller.

Fig. 11 shows an example of a typical domain network configuration. As shown in Fig. 11, domain network configurations are often complicated by many components such as computers, users, and servers. For the sake of simplicity, however, our experiment is performed with the network configuration shown in Fig. 12. The configuration of Fig. 12 is a simplified one of the configuration of Fig. 11. In the test network, for the agents to obtain the administrative privilege to the domain controller, it is necessary to move between computers twice and between users twice. Specifically, the process is as follows.

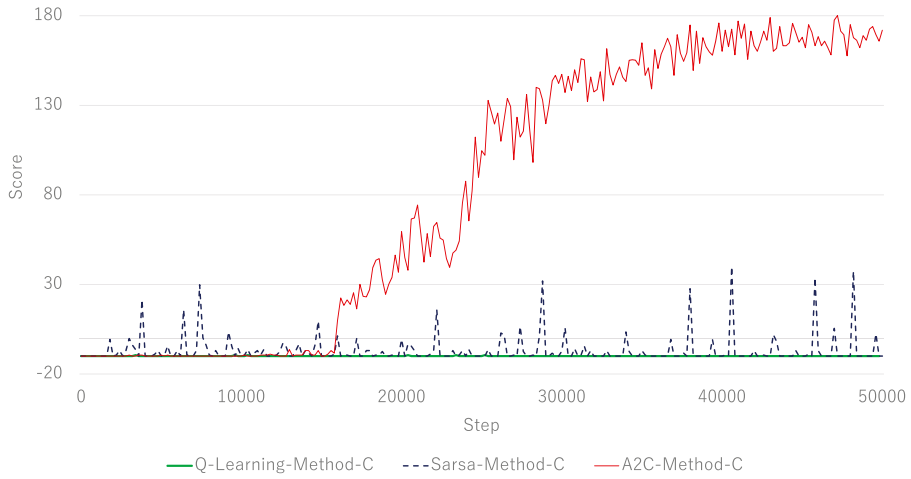

Fig. 9 – The transition of the gained rewards in the last 10 steps of each reinforcement learning model when C is applied.

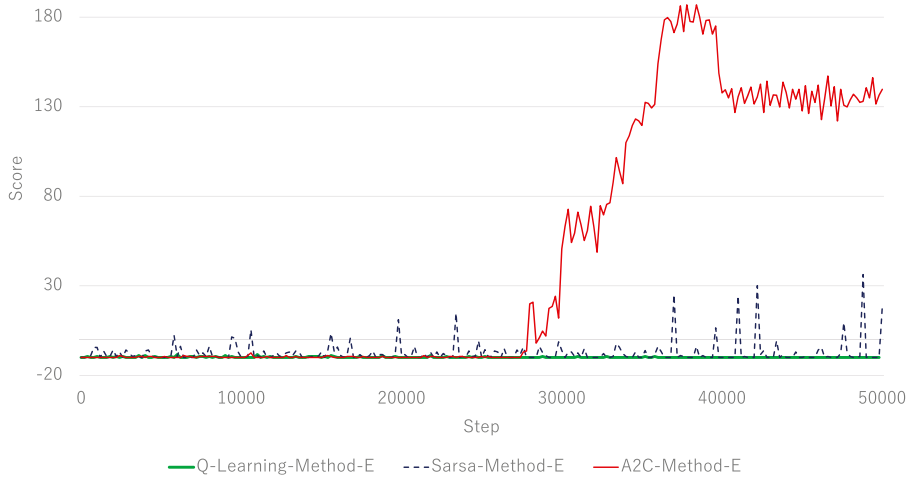

Fig. 10 – The transition of the gained rewards in the last 10 steps of each reinforcement learning model when E is applied.

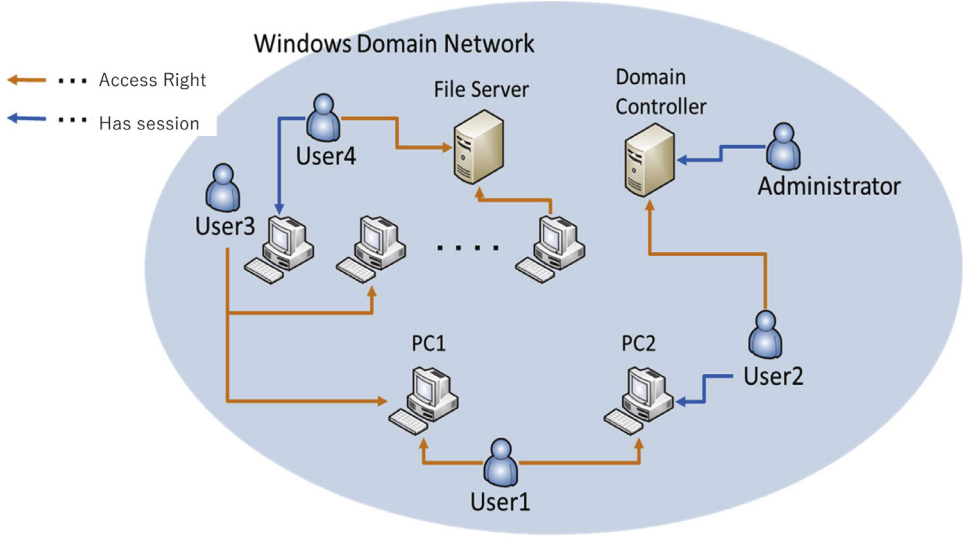

Fig. 11 – Typical domain network configuration example.

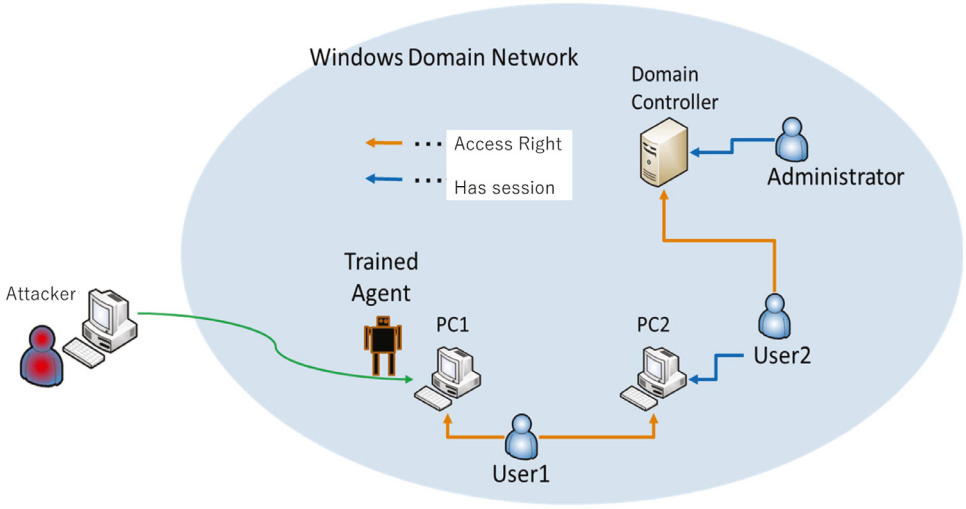

Fig. 12 – Test Windows domain network configuration. The agents start the action from PC1.

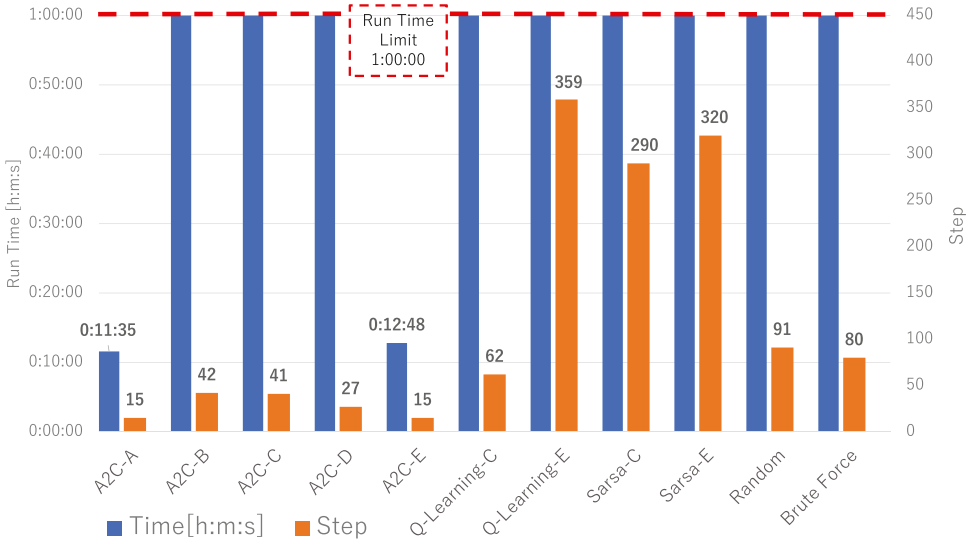

Fig. 13 – The run time and the number of execution steps for the each agent. 1 h is set as the upper limit time of the run time.

First, the agent moves from PC1 to PC2 with the permission of USER1. Second, the agent compromises the USER2 account on PC2. Third, the agent accesses the domain controller using the privileges of USER2. Finally, the agent compromises the Administrator account with administrative privilege to the domain controller.

Fig. 13 shows the number of execution steps and the run time for the each agent. Random in Fig. 12 represents the agent that randomly executes a selectable action. Similarly, Brute Force in Fig. 12 represents the agent that attempts brute force with the selectable actions. These two basic methods do not use reinforcement learning algorithms. Therefore, these are used as comparison targets to evaluate the effectiveness of applying reinforcement learning. The following results were obtained. First, we observed several significant differences in the run time. A2C-A and A2C-E achieved the goal in 15 steps within the time limit. Besides, A2C-A had the same number of steps as A2C-E, nevertheless, A2C-A had a slightly shorter execution time. In contrast, the other methods, including the basic methods, failed to achieve the goal within the time limit. Secondly, we observed several significant differences in the number of execution steps. The basic methods have no bias in choosing actions. Therefore, the average execution time of each step can be obtained from the number of steps of the basic methods; the average execution time of each step was 40 to 45 seconds. However, Q-Learning-E and all SARSA-Methods executed the steps approximately 3–4 times the average number of steps within the time limit. This indicates that the agents selected only actions with a short execution time. Finally, an interesting part of the results is the behavior of A2C-B and A2C-C in the test network. In the learning phase, A2C-B and A2C-C obtained higher rewards than A2C-A, nevertheless, only A2C-A achieved the goal in the test network. As mentioned above, these methods are different in the method of setting the success probability.

6. Discussion

6.1. *Effectiveness of deep reinforcement learning*

The purpose of this study is 1) to automate the post-exploitation, 2) to apply deep reinforcement learning to the post-exploitation as the method of the automation, and 3) to verify its effectiveness.

The first point we need to discuss is the results of the learning phase experiments. The results of Section 5.1 show that C and E could learn the behaviors to achieve significant lateral movement. In addition, the results show that A2C has better learning efficiency than Q-learning and SARSA. This suggests that A2C is suitable for the learning the task of the post-exploitation. The tabular approach is not suitable for this scenario because there are too many combinations of states and actions.

The point we must consider next is the result of the testing phase experiments. The results of Section 5.2 show that A2C-A and A2C-E could obtain the domain controller administrative privilege. This suggests that deep reinforcement learning could be applied to the post-exploitation and automate the task of the post-exploitation. Besides, our method achieved the goal in 15 steps. By contrast, the agentless automation models (Random and Brute Force) could not achieve the goal. It should be noted that the agents learned the appropriate behavior for their state so that compromise the domain controller as soon as possible. As a result, the agents followed the shortest path (not intentionally looking for the shortest path). In our experiments, there are cases that the agents cannot go through the path despite they could go through it before. This is because the success of the lateral movements depends on the success probability. Therefore, the shortest path search is difficult. If the training environment is static, we can find the shortest path. In this case, there are more efficient algorithms for the findings of the shortest path.

Here, we must consider the pros and cons of the agent-based and agentless automation models for a discussion of the effectiveness of the deep reinforcement learning. Table 4 outlines the pros and cons of each model. The result of the experiments shows that the agent-based automation model has advantages in terms of the efficiency of the post-exploitation. The agentless automation models will not be able to achieve their goals in a complex environment like our testing environment. This is the cons of the agentless automation model. On the other hand, the agent-based automation models require training, whereas the agentless automation models do not require it. This is the cons of the agent-based automation model since the proposed solution has the biggest problem with the preparation of the training sample. The agentless automation models that do not require the training do not have this problem. This is the pros of the agentless automation model.

Finally, let us consider the result of the failures in the testing phase of A2C-B and A2C-C. In A2C-B and A2C-C, the executed lateral movement always succeeds with a high probability. Therefore, the action selection may be biased toward the action that can be found first and obtained the reward. They are considered that overfitted for the training environment and failed on the test network.

6.2. *Comparison*

DeepExploit (Isao) is a practical framework that applies reinforcement learning to cybersecurity scenarios. We should compare Deep Exploit with the proposed solution carefully since they have some common points but also have major differences.

Table 5 summarizes the comparison of our solution and DeepExploit. A common feature of the proposed solution and DeepExploit is to use deep reinforcement learning as an automated approach. Furthermore, the reinforcement learning models they use are similar (A2C and A3C). Meanwhile, as we described in Section 2.2, automation targets of Deep Exploit are the initial exploitation of cyber-attacks. DeepExploit performs a port-scanning on the target server and executes the best exploit for the target service: This is the first step of the cyber-attack. By contrast, the proposed solution automates the steps after DeepExploit ran, that is, the post-exploitation. The post-exploitation task involves state transitions, unlike initial exploitation. For example, in the case of the DeepExploit, services on the port of the target server define the state of agents. Therefore, the actions of the agent do not change the state. In the case of the proposed solution, for example, if the agent succeeds in the movement to another computer, it needs to take the following actions accordingly; therefore, the state transition should be represented. Summary, the application of reinforcement learning to cybersecurity scenarios that require state transitions is an advantage of the proposed solution. Another difference is the training sample of the agents. DeepExploit trains agents from real-world samples, while the proposed solution trains the agents from the synthetic samples (The reason for using the synthetic sample is described

Table 4 – Comparison of agent-based model and agentless model.

	Pros	Cons
Agent-based Model	efficient flexible	need training need training environment
Agent-less Model	not need training	inefficient impractical inflexible

Table 5 – Comparison of the proposed solution and DeepExploit.

	Reinforcement learning model	Target for automation	State transition
DeepExploit	A3C	Exploitation (initial exploitation)	None
Proposed solution	A2C	Post-exploitation	Represented

in Section 6.4.). In this regard, DeepExploit is a more realistic framework than the proposed solution.

Next, we must consider comparison with studies in which reinforcement learning was applied to the cyber-security simulation. Our work demonstrated the applicability of reinforcement learning in the actual environment. The previous works applied reinforcement learning to the cyber-security-simulation (Elderman et al., 2017; He et al., 2016). However, these works have demonstrated the applicability of reinforcement learning only in the simulation environment. Specifically, the previous works do not support automation such as lateral movement and privilege escalation in an actual environment. By contrast, our work has demonstrated it in the actual environment. We implemented the reinforcement learning agents to be executable in the actual environment. This paper demonstrated the practical automation of lateral movement in the actual environment. Altogether, our work contributed to improving the practicality of the previous works.

Finally, let us discuss the comparison with the agentless model, DeathStar (Deathstar). DeathStar is a Python script that uses Empire's RESTful API to automate gaining Domain Admin rights in Active Directory environments using a variety of techniques. DeathStar and our method are similar in that they rely on PowerShell Empire and automate the post-exploitation. However, DeathStar is just a script. It does nothing more than its programmed behavior. Altogether, DeathStar has the pros and cons of the agentless model mentioned in Section 6.1.

6.3. *Research ethics*

Our method is an effective and proactive approach to cyber-threats. Our method simulates the attack methods that an attacker actually uses in a real environment. This makes easy the red team test which requires high cost and highly skilled personnel, that is, this can improve the defense. However, it is necessary to take preventive measures against the abuse of the research on attack techniques. The PowerShell Empire has been deprecated support due to abuse cases. Besides, the implementation code of this research using the PowerShell Empire is not published to prevent abuse.

6.4. *Limitations*

As we mentioned in Section 6.3, the PowerShell Empire Framework is no longer maintained. Since the PowerShell Empire modules are defined as the agent's action, it is impossible to update and improve the agent's action unless the PowerShell Empire is not updated. Thus, the drawback of using the PowerShell Empire is that the implementation of the proposed method depends on the PowerShell Empire; however, it can replace with other frameworks.

Besides,the following points are left as future problems.

There are little diversity in the training environment. Our method diversified the training environment by setting the success probability of the agent's action: 5 success probability setting patterns were evaluated. However, it will be difficult to represent the diversity of the actual environment only by the success probability. The methods to diversify the learning environment are the subject of a future study.

Besides, in this paper, we attempted only one method of agent state definition. Depending on the method of the state definition, the agent's behavior and the learning progress may change. Thus, exploring and evaluating other methods to define the state of the agent is also the subject of a future study.

Finally, the samples used to train the agents are synthetic samples and the real-world testing is conducted in the limited condition. Ideally, the agent should learn the exploitations from real-world samples rather than synthetic samples. This problem limits the effectiveness of agent training and learning. Specifically, the agents will probably not be able to adapt to a wide variety of characteristic environments at this time. Therefore, in this regard, our contribution is that it demonstrated the potential of a planned (or modeled) cyber-attack optimized by deep reinforcement learning.

7. Conclusion and future work

This paper proposes a method to automating the post-exploitation with deep reinforcement learning and evaluated the effectiveness in the actual environment. The reinforcement learning agents are implemented to be executable in the actual environment. In our experiments, we compared the methods of A2C, which are our methods, with the methods of Q-Learning, SARSA, and 2 basics. The results show the superiority of our method. Specifically, our method could obtain reward most efficiently and obtain the administrative privileges to the domain controller. This is the first study, to the best of our knowledge, that automates post-exploitation by reinforcement learning in the actual environments. Interesting directions for future work include 1) studying the methods to diversify the learning environment, 2) designing and evaluating other definitions of the agents state.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Ryusei Maeda: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing - original draft, Visualization. **Mamoru Mimura:** Conceptualization, Data curation, Supervision, Project administration, Funding acquisition.

REFERENCES

Apruzzese G, Colajanni M, Ferretti L, Guido A, Marchetti M. On the effectiveness of machine and deep learning for cyber security. In: 10th International Conference on Cyber Conflict, CyCon 2018, Tallinn, Estonia, May 29, - June 1, 2018; 2018. p. 371–90. doi:10.23919/CYCON.2018.8405026.

B.Deply, 2014. Mimikatz. https://github.com/gentilkiwi/mimikatz.

Bland JA, Petty MD, Whitaker TS, Maxwell KP, Cantrell WA. Machine learning cyberattack and defense strategies. Computers & Security 2020;92:101738. doi:10.1016/j.cose.2020.101738.
Boileau, A.,. Trust transience: Post intrusion ssh hijacking. In Blackhat Briefings August 2005.
Çavusoglu Ü. A new hybrid approach for intrusion detection using machine learning methods. Appl. Intell. 2019;49(7):2735–61. doi:10.1007/s10489-018-01408-x.
Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: synthetic minority over-sampling technique. J. Artif. Intell. Res. 2002;16:321–57. doi:10.1613/jair.953.
Cui Z, Xue F, Cai X, Cao Y, Wang G, Chen J. Detection of malicious code variants based on deep learning. IEEE Trans. Ind. Inf. 2018;14(7):3187–96. doi:10.1109/TII.2018.2822680.
Deathstar, https://github.com/byt3bl33d3r/DeathStar.
Duckwall, S., Campbell, C.,. Hello, my name is microsoft and I have acredential problem in blackhat USA 2013 white papers 2013. https://media.blackhat.com/us-13/US-13-Duckwall-Pass-the-Hash-WP.pdf.
Dunagan J, Zheng AX, Simon DR. Heat-ray: combating identity snowball attacks using machinelearning, combinatorial optimization and attack graphs. In: Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11–14, 2009; 2009. p. 305–20. doi:10.1145/1629575.1629605.
Elderman R, Pater LJJ, Thie AS, Drugan MM, Wiering M. Adversarial reinforcement learning in a cyber security simulation. In: van den Herik HJ, Rocha AP, Filipe J, editors. In: Proceedings of the 9th International Conference on Agents and Artificial Intelligence, ICAART 2017, Volume 2, Porto, Portugal, February 24–26, 2017.. SciTePress; 2017. p. 559–66. doi:10.5220/0006197105590566.
Ghanem MC, Chen TM. Reinforcement learning for intelligent penetration testing. In: 2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4); 2018. p. 185–92. doi:10.1109/WorldS4.2018.8611595.
He X, Dai H, Ning P. Faster learning and adaptation in security games by exploiting information asymmetry. IEEE Trans. Signal Processing 2016;64(13):3429–43. doi:10.1109/TSP.2016.2548987.
Isao, T.,. Metasploit meets machine learning. https://www.mbsd.jp/blog/20180228.html.
Jia J, Gong NZ. Attriguard: a practical defense against attribute inference attacks via adversarial machine learning. In: 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15–17, 2018; 2018. p. 513–29.
Kim T, Kang B, Rho M, Sezer S, Im EG. A multimodal deep learning method for android malware detection using various features. IEEE Trans. Inf. Forensics Secur. 2019;14(3):773–88. doi:10.1109/TIFS.2018.2866319.
Lah AAA, Dziyauddin RA, Azmi MH. Proposed framework for network lateral movement detection based on user risk scoring in SIEM. In: 2018 2nd International Conference on Telematics and Future Generation Networks (TAFGEN); 2018. p. 149–54. doi:10.1109/TAFGEN.2018.8580484.
Leen, T. K., Dietterich, T. G., Tresp, V. (Eds.), 2001. Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA, MIT Press.
Milosevic N, Dehghantanha A, Choo KR. Machine learning aided android malware classification. Computers & Electrical Engineering 2017;61:266–74. doi:10.1016/j.compeleceng.2017.02.013.
Mnih V, Badia AP, Mirza M, Graves A, Lillicrap TP, Harley T, Silver D, Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19–24, 2016; 2016. p. 1928–37.
Powershell empire, | building an empire with powershell. https://www.powershellempire.com/.
Rapid7,. Metasploit | penetration testing software, pen testing security | metasploit. https://www.metasploit.com/.
Sharma A, Kalbarczyk Z, Barlow J, Iyer RK. Analysis of security data from a large computing organization. In: Proceedings of the 2011 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2011, Hong Kong, China, June 27–30 2011; 2011. p. 506–17. doi:10.1109/DSN.2011.5958263.
Shadowmove. In: 29th USENIX Security Symposium (USENIX Security 20). A stealthy lateral movement strategy. Boston, MA: USENIX Association; 2020.
Simard PY, LeCun Y, Denker JS, Victorri B. Transformation invariance in pattern recognition: tangent distance and propagation. Int. J. Imaging Systems and Technology 2000;11(3):181–97. doi:10.1002/1098-1098(2000)11:3<181::AID-IMA1003>3.0.CO;2-E.
Tian Z, Shi W, Wang Y, Zhu C, Du X, Su S, Sun Y, Guizani N. Real-time lateral movement detection based on evidence reasoning network for edge computing environment. IEEE Trans. Industrial Informatics 2019;15(7):4285–94. doi:10.1109/TII.2019.2907754.
Wu, Y., Mansimov, E., Liao, S., Radford, A., Schulman, J.,. Openai baselines:acktr & a2c. https://openai.com/blog/baselines-acktr-a2c/.
Wueest, C.,. The increased use of powershell in attacks. https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/increased-use-of-powershell-in-attacks-16-en.pdf.

Ryusei Maeda received his B.E. in Engineering from National Defense Academy of Japan, in 2020. Currently, he is a member of the Japan Maritime Self-Defense Force.

Mamoru Mimura received his B.E. and M.E. in Engineering from National Defense Academy of Japan, in 2001 and 2008 respectively. He received his Ph.D in Informatics from the Institute of Information Security in 2011 and M.B.A. from Hosei University in 2014. During 2001–2017, he was a member of the Japan Maritime Self-Defense Force. During 2011–2013, he was with the National Information Security Center. Since 2014, he has been a researcher in the Institute of Information Security. Since 2015, he has been with the National center of Incident readiness and Strategy for Cybersecurity. Currently, he is an Associate Professor in the Department of C.S., National Defense Academy of Japan.