applied sciences

MDPI

Article

Cyber Attack and Defense Emulation Agents

Jeong Do Yoo [1], Eunji Park [1], Gyungmin Lee [1], Myung Kil Ahn [2], Donghwa Kim [2], Seongyun Seo [2] and Huy Kang Kim [1,*]

[1] Graduate School of Information Security, Korea University, 145 Anam-ro, Seongbuk-gu, Seoul 02841, Korea; opteryx25104@korea.ac.kr (J.D.Y.); epark911@korea.ac.kr (E.P.); pinjk123@korea.ac.kr (G.L.)

[2] Agency for Defense Development, Seoul 05771, Korea; happyahn@add.re.kr (M.K.A.); dhkim@add.re.kr (D.K.); syseo@add.re.kr (S.S.)

* Correspondence: cenda@korea.ac.kr

Received: 27 February 2020; Accepted: 18 March 2020; Published: 21 March 2020

Abstract: As the scale of the system and network grows, IT infrastructure becomes more complex and hard to be managed. Many organizations have a serious problem to manage their system and network security. In addition, vulnerabilities of hardware and software are increasing in number rapidly. In such a complex IT environment, security administrators need more practical and automated threat assessment methods to reduce their manual tasks. Adversary emulation based automated assessment is one of the solutions to solve the aforementioned problems because it helps to discover the attack paths and vulnerabilities to be exploited. However, it is still inefficient to perform the adversary emulation because adversary emulation requires well-designed attack scenarios created by security experts. Besides, a manual-based penetration test cannot be frequently performed. To overcome this limitation, we propose an adversary emulation framework composed of the red team and blue team agent. The red team agent carries out automated attacks based on the automatically generated scenarios by the proposed framework. The blue team agent deploys defense measures to react to the red team agent's attack patterns. To test our framework, we test multiple attack scenarios on remote servers that have various vulnerable software. In the experiment, we show the red team agent can gain an administrator's privilege from the remote side when the blue team agent's intervention is not enabled. The blue team agent can successfully block the red team's incoming attack when enabled. As a result, we show our proposed framework is beneficial to support routine threat assessment from the adversary's perspective. It will be useful for security administrators to make security defense strategy based on the test results.

Keywords: adversary emulation; agent modeling; automated agent; blue-teaming; red-teaming

1. Introduction

With the development of information technology and the growing scale of system and network, the threats against computer security are being increased. According to the report of 'HACKMAGEDDON' and the statistics from CVE-detail, the number of security incidents is continuously increasing [1,2]. It implies security administrators need to find out the vulnerabilities that reside in their organizations and develop countermeasure to confront the increasing cyber-threats. To this end, adversary emulation is regarded as one of the efficient ways for threat assessment and vulnerability management.

Adversary emulation (or adversary simulation) is an activity to respond to highly sophisticated attacks organizations might face [3]. Adversary emulation has two main actors, the red team and the blue team. A red team is an actor that performs cyber-attacks in the emulation, and it attacks the target system and network using various attack techniques. In the middle of the attack process,

the red team can also check whether some critical configuration files are well set. On the opposite side, a blue team is an actor of cyber-defense in the emulation. It defends the system and network against attackers by preventing, detecting, and responding to attacks. In general, adversary emulation is performed in a virtual environment that has the same configurations of the target server and network. A test in a virtual environment can prevent possible side-effects (e.g., application error, system panic, or unexpected rebooting) during the penetration test. Although many state-of-the-art technologies such as server and network virtualization can support the adversary emulation, it still requires cybersecurity experts to do a lot of manual labour. These manual works include creating test scenarios, provisioning server and network resources to be tested and writing exploiting scripts. These kinds of works need a person who has expert knowledge in security as well as computer science and engineering. It can be a limitation when an organization considers conducting a routine security threat assessment using the adversary emulation with security experts. Therefore, the need for adversary emulation by the automated system has been increasing to support frequent security assessments and to minimize the manual work by experts.

The remainder of this paper is organized as follows. Several related projects and research on adversary emulation are reviewed in Section 2. We describe our proposed model in Section 3. The main component of our model, red team agent and blue team agent are described in Sections 4 and 5. In Section 6, we show the feasibility of the agents through three scenario-based experiments. In Section 7, we discuss several issues of the proposed agent. Finally, we summarize our work and conclude in Section 8.

2. Related Works

There are several tools for the automated adversary emulation: CALDERA [4], CASCADE [5], Uber Metta [6], Atomic Red Team [7], and APTSimulator [8]. The CALDERA is developed by MITRE corporation and it automatically attacks target systems using tactics and techniques described in the ATT&CK framework [9]. Applebaum et al. [10] introduced the design and implementation of CALDERA. Miller et al. [11] described how to handle the planning problem of adversary emulation by CALDERA, when CALDERA needs to handle uncertainties of the target systems. The CASCADE automatically analyzes suspicious behaviors and determines their maliciousness with the ATT&CK framework. It mainly focuses on the analysis of the detected threats and not on threat mitigation.

The Metta is a tool for adversary emulation using Redis/Celery, Python and Vagrant. The Metta uses the concept of 'Action', which is a sequence of commands. Metta can execute the defined action when it performs the adversary emulation. Metta supports creating a batch of actions to make scenarios for adversary emulation. Metta's such functionalities support to create a test scenario and actions without difficulty. It is very helpful to security administrators but its scenario creation process is not automated.

The Atomic Red Team is an execution framework using PowerShell, Python and Ruby. 'Atomic test' is a unit of the Atomic Red Team and it represents the sequence of commands for a behavior. Each atomic test can run on multiple operating systems including Linux, Windows, and macOS. Although the Atomic Red Team has many merits to test various systems, it does not have the concept of a scenario. It cannot execute a series of atomic tests automatically. Thus, users need to write a batch job with scripting language manually when they want to create a test scenario.

The APTSimulator is another adversary emulation tool that can run through a Windows batch file.The APTSimulator uses the concept of 'test-set', a sequence of Windows commands. APTSimulator launches the 'test-set' to carry out the adversary emulation. As well as the Atomic Red Team, it does not support the automated scenario test (i.e., series of test-sets).

In addition to the several tools described above, there are also related studies on the cyber-attack simulation and modeling. In Kavak et al.'s work [12], the authors focused on the simulation environment for building the simulation scenario. In this study, characterizations of the system and actor were

addressed. One consideration is that the attacker and defender's behavior modeling was not covered in depth.

In the work of Kuhl et al. [13], the cyber-attack modeling was built for the network security analysis covering the both external and internal network. The study categorized all available cyber-attacks into 10 stages and used them with graph-based approach. It assumed the attacker had full information about the target's environment, thus the optimal decision for the cyber-attack could be gained.

In the work of Applebaum et al. [14], the authors designed the emulation testbed. Their testbed established the attacker's modeling on the Windows enterprise network environment. Multiple experiments were conducted on the adversary emulation and the results were compared. The experiments used various strategies for automated attacks, such as random selection, perfect knowledge, and so on.

3. Proposed Model

There have been many studies on adversarial emulation. Adversarial emulation is practical to test the real security level of organizations. However, previous studies have several limitations in common. Attackers need to develop new attack patterns to find another penetration way. If they fail to develop new ones, then the routine test is not meaningful because the result will be the same as the previous test result. Defenders need to update the defending system continuously. They need to check the possible new attack path whenever new vulnerabilities or attack patterns are revealed. Such natures of the defense processes make the defense repetitive and time-consuming. For these reasons, the adversary emulation by the automated system can be a solution. However, many adversarial emulation programs still do not support a whole automated process from the scenario generation to defense measure deployment.

To overcome the limits of the emulation, we propose a cyber-attack and defense emulation model and we implement an adversary emulation framework. Our proposed model is designed to achieve two goals: supporting routine threat assessment and helping blue teams to make security defense strategies. The primary goal of the adversary emulation is to provide the most probable attack for the routine threat assessment using known security vulnerabilities. To this end, we design the red team agent. The red team agent tries to find security vulnerabilities from the targets to derive diversified attack scenarios. The red team agent performs a lot of iterations based on the scenarios to scan existing vulnerabilities, and it manages a series of attacks in order based on the attack scenarios. Another goal is to enhance security administrators' defense skills. We design the blue team agent that can deploy defense measures to react to the red team agent's attack patterns. During the emulation process, security administrators can learn various defense skills by engaging the incoming attacks from the red team agent. The blue team agent's defense performance configured by security administrators can be a defense capability of the organization.

4. Red Team Agent

This section presents the overall architecture of the red team agent and how it executes automated attacks. For the automated attacks, the agent should be aware of its state at first. Then, the agent should plan and carry out attacks on the basis of the current state. Inspired by STRIPS, a well-known automated planning method, we defined two primary concepts: 'state' and 'attack technique.' The state is a data structure including the description of the red team agent's current status against a target system (e.g., IP address, port list, and gained privilege). The state information makes the red team agent understand the current situation. The attack technique is a unit of attacks for the red team agent and it has pre-conditions and post-conditions. The red team agent selects attack techniques having the high probability of a success based on the state information. Thus, 'the red team agent attacks' implies the agent uses one of the attack techniques to exploit the target. The overall process of the automated attack by the red team agent is described in Section 4.4.

4.1. Overall Architecture of Red Team Agent

Figure 1 depicts the red team agent's structure. The red team agent is composed of 'agent master', 'technique DB', 'scenario generator', 'listener', and 'state'.

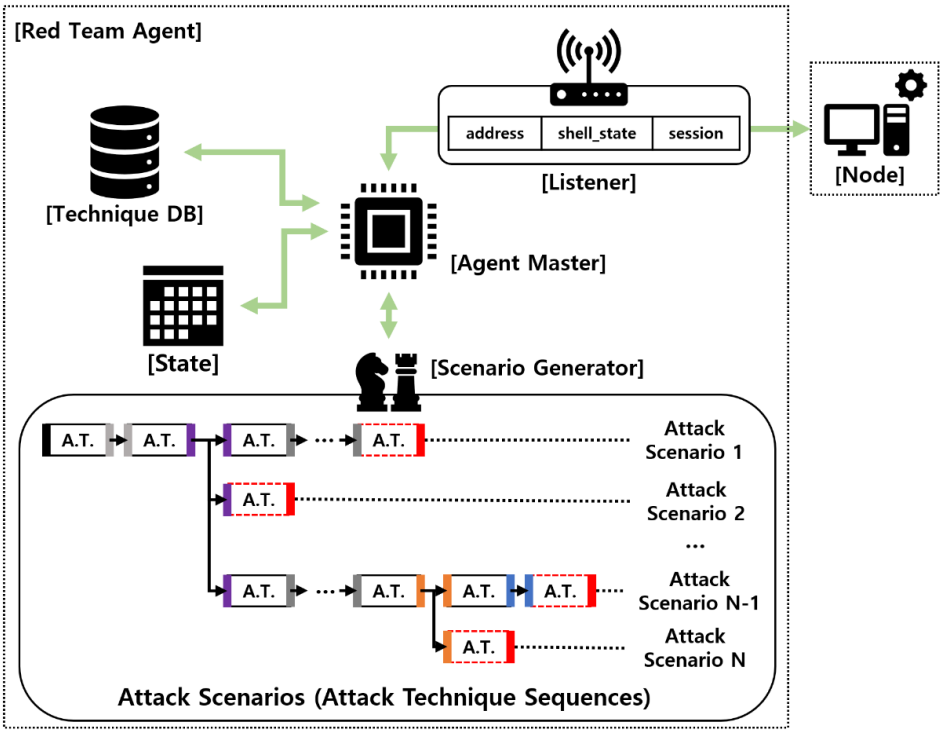

Figure 1. Overall architecture of red team agent.

4.1.1. Agent Master

Agent master manages and controls the other components.

4.1.2. Technique DB

'Technique DB' is a repository of attack techniques. It has four fields for each attack technique: ID, pre-conditions, post-conditions, and commands. The fields are described in detail in Section 4.3. The pre-conditions and post-conditions of each attack are described in the database. The Technique DB helps the red team agent generate attack scenarios along with the red team agent's state.

4.1.3. Scenario Generator

'Scenario generator' creates attack scenarios. An attack scenario is a sequence of attack techniques that are selected to achieve the red team agent's goal (usually, gaining administrator's privilege). Thus, the scenario generator queries the technique DB to select attack techniques, and then compose them to derive the most probable attack sequence. After the scenario generator generates a scenario, the red team agent executes attack techniques following the order described in the scenario.

4.1.4. Listener

'Listener' is a component that manages gained shells of a target. Once shells of the target are gained, the listener maintains the established connections to keep them alive. Shell connection reversing or port-forwarding techniques are commonly used for this purpose.

However, the established connection is temporal and unstable in most cases. For example, the target system's administrator can kill the connection, or the target system can be rebooted unexpectedly. To gain permanent access to the target, a remote access tool (RAT) can be used.

The listener has a table containing 'IP address', 'state', and 'session' information in order to manage the states of shells and sessions.

4.1.5. State

'State' is a set of key-value pairs which contains important information of the red team agent. There are nine fields for the state and each of them has paired information as follows: (Address: IP address), (Protocol: protocol used in the port), (OS: Operating System info.), (Ports: list of open port), (Shell: 'TEMPORAL', 'DOWNLOADED', 'PERMANENT'), (Privilege: superuser, regular user), (Binaries: list of binaries installed on the target node), (CWD: current working directory), (Misudo: exploitable sudo configurations). Section 4.2 explains them in detail.

4.2. Red Team Agent's State

The state represents the information of the red team agent performing attacks. Each state corresponds to a single node (system) under the current attack. Table 1 depicts the fields (keys) of the state.

- 'Address' is an address of the target node. The address has a form of 'x.x.x.x.'
- 'Protocol' is a protocol to access the target node. The most common protocol is IPv4.
- 'OS' represents the operating system installed on the target node. The red team agent executes an attack technique by inputting its 'Commands' field. Because commands to be sent are different according to the operating system. Thus, identifying the target system's OS is essential. Using the identified OS information, the red team agent can list up the possibly exploitable vulnerabilities in advance.
- 'Ports' is a list of port numbers. These ports will be the access point to the target node by the red team agent. The list contains the opened-port numbers, and the list can have one or more open ports if the target server runs one or more services. This field is valid only if 'Protocol' is IPv4 or IPv6.
- 'Shell' indicates the state of shells. Each shell can have one of three values: 'TEMPORAL', 'DOWNLOADED', and 'PERMANENT'. 'TEMPORAL' is the state that the red team agent temporally takes the current shell. If the connected process is terminated, the red team agent loses the shell. 'DOWNLOADED' is the state that the red team agent has sent a binary such as RAT to the target node to acquire the shell permanently. 'PERMANENT' is the state that the red team agent has acquired the permanent shell after DOWNLOADED state is over. Even if binaries like RAT are terminated, they would be consistently rebooted to make the red team agent secure the shell to access the target node. For instance, using 'cron', a job scheduler in Unix-like OS, it is possible to set the installed binaries to be rebooted periodically in case they get terminated.
- 'Privilege' is a level of acquired authority over the target node. There are two types of privilege. One is an administrative authority (superuser) which is the highest level of granted permission. The other type is a regular authority (regular user). If the red team agent gained only the regular authority, the behavior is restricted in the target node.
- 'Binaries' is a list of binaries (executable programs) installed on the target node. It can be further used after the red team agent has successfully intruded into the target node.
- 'CWD' is a current working directory of the shell. Based on this information, the red team agent can know an absolute path and a relative path of the file system on the target node.
- 'Misudo' is an array of exploitable sudo configurations. For instance, 'sudofind' can be an element of this array, because it makes 'find' command be executed without the root password. The red team agent can determine if the target node has any sudo related misconfiguration.

The state is used to verify the pre-conditions of each attack technique. The red team agent can identify what attack techniques it can use. The red team agent updates the state after using each attack technique referring to their post-conditions. After the attack successfully works, the red team agent acquires the higher authority. Then the 'Privilege' field of the state changes from the regular user to the superuser.

The state is also used to judge the red team agent's goal against the target node. Let us assume that the objective of the red team agent is "obtaining a permanent shell and an administrative authority". Then, the red team agent attacks the target node until the following statement of the state is fulfilled: *Shell == PERMANENT && Privilege == ADMINISTRATIVE && Address == Target_Node's_Address*.

Table 1. Model of red team agent's state.

Field	Description
Address	Address of the target node
Protocol	Protocol to access the target node
OS	Operating System installed in the target node
Ports	Ports to access the target node
Shell	State of the shell to access the target node (TEMPORAL or DOWNLOADED or PERMANENT)
Privilege	Privilege over the target node
Binaries	Available binaries installed in the target node
CWD	Current working directory of the shell which have accessed to the target node
Misudo	Array of exploitable sudo configurations

4.3. Attack Technique

The red team agent uses attack techniques to conduct automated attacks. For automated attacks, the red team agent should decide what attack technique is available on the current state and it also should predict the result of using the selected technique.

Table 2 shows that each technique has four fields: 'technique ID', 'pre-conditions', 'post-conditions', and 'commands'. The concept of pre-conditions and post-conditions was inspired by STRIPS [15] and MITRE Caldera [10].

- Technique ID is the identifier of attack techniques.
- Pre-conditions are requirements of an attack technique. Each pre-condition consists of one or two operands and an operator. A field of the red team agent's state can be an operand, and there are four operators: EQUAL, EXIST, NON-EXIST, and INCLUDE. As an example of a pre-condition, 'Address EQUAL "192.168.10.1"' means 'the red team agent's state says Address is equal to "192.168.10.1".'
- Post-conditions represent the conditions which are satisfied after an attack technique is successfully used. Combined with the pre-conditions and the post-conditions, attack techniques can be linked with each other. The red team agent searches the technique DB to find the next coming attack technique, checking if the pre-condition of the attack technique is completely fulfilled by the post-condition of the current state.

  We introduce 'UNKNOWN,' a keyword used in post-conditions, to represents an undetermined value. Some post-conditions can include the keyword; The keyword is necessary to deal with uncertain information during scenario generation.

  For example, we assume that the installed OS in the target system is either Windows or Linux. In this case, the scenario generator cannot process further until the OS information is exactly classified. Therefore, to continue the attack scenario generation, the UNKNOWN keyword should fill in uncertain elements of post-conditions. If a selected technique has a post-condition, 'OS is UNKNOWN,' then the following technique contains the pre-condition saying 'OS is Windows' or 'OS is Linux.' When the red team agent performs attack scenarios, if it can take certain information, then the UNKNOWN value of the post-condition can be replaced with the acquired information.

  Admittedly, some attack scenarios can be discarded during the scenario performance process. As mentioned above, attack techniques having 'OS is Windows' or 'OS is Linux' as their pre-conditions satisfy the 'OS is UNKNOWN' condition. If the determined OS information is 'OS is Linux', then the scenarios with attack techniques only available on Windows should be discarded. Instead, other attack scenarios with techniques available on Linux are adopted.

- 'Commands' is a list of file paths that contain commands for the attack techniques. The red team agent inputs the commands sequentially to operate the attack techniques in order.

Table 2. Model of the attack technique.

Field	Description
Technique ID	Technique identifier
Pre-conditions	Requirements to use the technique
Post-conditions	Satisfactions after using the technique
Commands	File paths, which include commands to use the technique

4.4. Overall Process and Algorithm

The overall process of the red team agent is depicted in Algorithm 1. The process can be divided into two major processes: scenario generation and scenario performance. The scenario generation selects the attack techniques to use and organizes them in a proper order. By the time the scenario generation is completed, multiple attack scenarios are drawn. Then, the process iterates on the scenarios and performs the attacks on each scenario. If it succeeds in performing the attacks, the iteration terminates with the 'Attack Succeeded' message. Otherwise, it continues to execute the next iteration of other scenarios. In case when all scenarios fail, the process terminates with the 'Attack Failed' message.

Algorithm 1: The overall process.

```
initialization();
scenarios ← generate_scenarios(...);
foreach scenario ∈ scenarios do
    result ← perform_scenario(scenario);
    if result is true then
        print("Attack Succeeded");
        return;
    end
end
print("Attack Failed");
```

Algorithm 2 shows the process of the scenario generation.The scenario generator compares the red team agent's state to the pre-conditions of each attack technique and selects available attack techniques. As the algorithm works recursively, it accumulates the attack techniques one after another along with the pre-conditions and the post-conditions. At this moment, it uses copied states instead of actual states to update post-conditions of the newly selected attack techniques. If the sequence of attack techniques leads the state to the red team agent's goal, then the sequence is appended to the list of valid attack scenarios. The process terminates after all scenarios based on the available unit attacks are drawn.

Algorithm 3 depicts the process of the scenario performance. The red team agent performs attacks using the attack scenarios generated in the previous process.For each attack scenario, it uses the techniques of the scenario one by one. The process aims to show the result of the attack.The result of the process is a failure in two cases. One case is when the state does not satisfy the pre-conditions of the following attack technique. The other is there is no more attack technique to execute in the loop and the goal has not been achieved. If the goal of the red team agent is achieved, the process terminates with the result of success.

Algorithm 2: generate_scenarios.

```
Input: state
Input: attacks (all available attack techniques)
Input: sequence (cumulative attack technique sequence)
Output: scenarios (available attack scenarios)
scenarios ← {};
if is_goal_achieved(state) then
    new_scenario ← make_scenario(sequence);
    scenarios ← scenarios + new_scenario;
    return scenarios;
end
next_attacks ← get_available_attacks(state, attacks);
foreach next_attack ∈ next_attacks do
    copied_state ← state;
    copied_attacks ← attacks;
    copied_sequence ← sequence;
    copied_state ← update_postcondition(copied_state, next_attack);
    copied_attacks ← copied_attacks - next_attack;
    copied_sequence ← copied_sequence + next_attack;
    result_scenario ← generate_scenarios(copied_state, copied_attacks, copied_sequence);
    scenarios ← scenarios + result_scenario;
end
```

Algorithm 3: The perform_scenario.

```
Input: scenario (attack sequence that can satisfy the goal)
state ← {};
foreach attack ∈ scenario do
    if is_goal_achieved(state) then
        return true;
    end
    if not is_precondition_satisfied(state, attack) then
        return false;
    end
    state ← perform_attack(state, attack);
end
return false;
```

4.5. Generating and Performing Attack Scenarios

4.5.1. Generating Attack Scenarios

This section shows one example of how the red team agent generates scenarios for the attack. The attack scenario generation is according to Algorithm 2. In the scenario generation, two key points need to be reminded. The first point is that the state in this phase is not the actual state of the red team agent, but a copy of the actual one. Using the copied state, the red team agent would generate attack scenarios by selecting available attack techniques.The other point is the keyword 'UNKNOWN' on some post-conditions. As mentioned in Section 4.3, the keyword is used to replace the uncertain information without halting scenario generation process.

Figures 2–7 depict one example of the scenario generation. Each figure consists of four factors: 'technique pool', 'pre-state', 'post-state', and 'technique sequence'. Technique pool is a collection of potentially available attack techniques that are not on the technique sequence yet. Pre-state is a state of the red team agent before one attack technique is selected from the technique pool. 'Pre-condition

validation' should be performed to select attack techniques suitable for the pre-state. Contrary to the pre-state, the post-state is a state of the red team agent after one attack technique is selected. Assuming the technique succeeds, the pre-state is updated along with the post-conditions of the selected attack technique and it becomes the post-state. This process is called the 'post-condition reflection.' 'Technique sequence' is a sequence of attack techniques in the selected order. Every time the sequence gets new attack techniques, 'goal verification' is performed to verify whether the post-state achieves the goal of the red team agent. If the result of the verification says the goal is achieved, the technique sequence will be considered as a valid scenario and be stored. Otherwise, the scenario generation will keep updating the technique sequence until the goal is achieved.

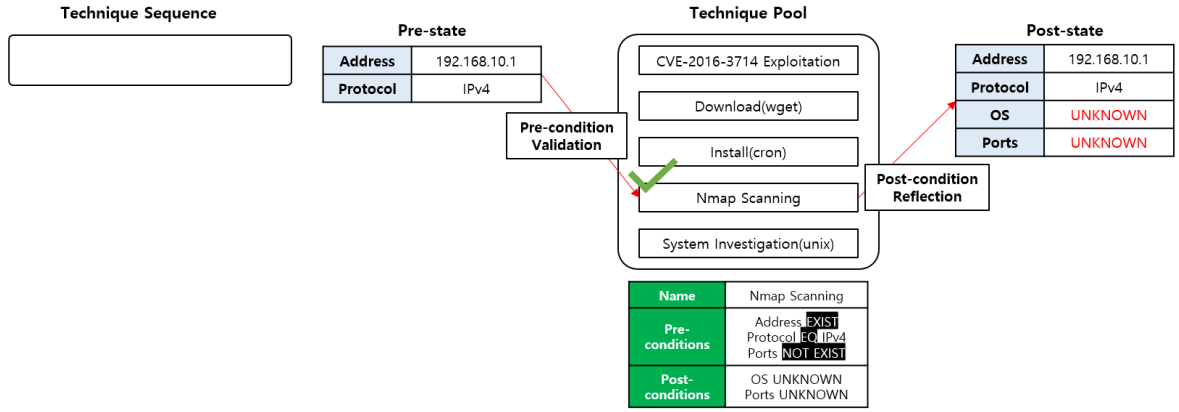

Figure 2. The example of the process of scenario generation: port scanning chosen.

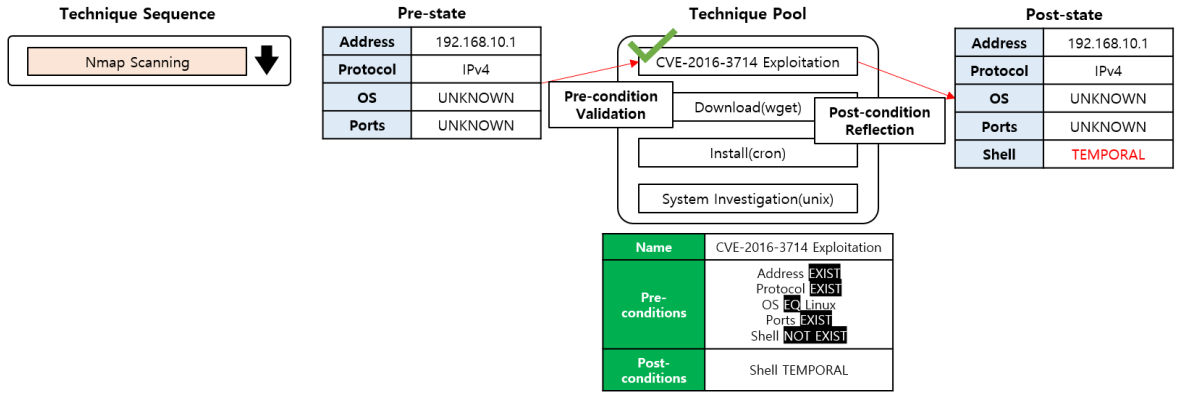

Figure 3. The example of the process of scenario generation: CVE-2016-3714 exploitation chosen.

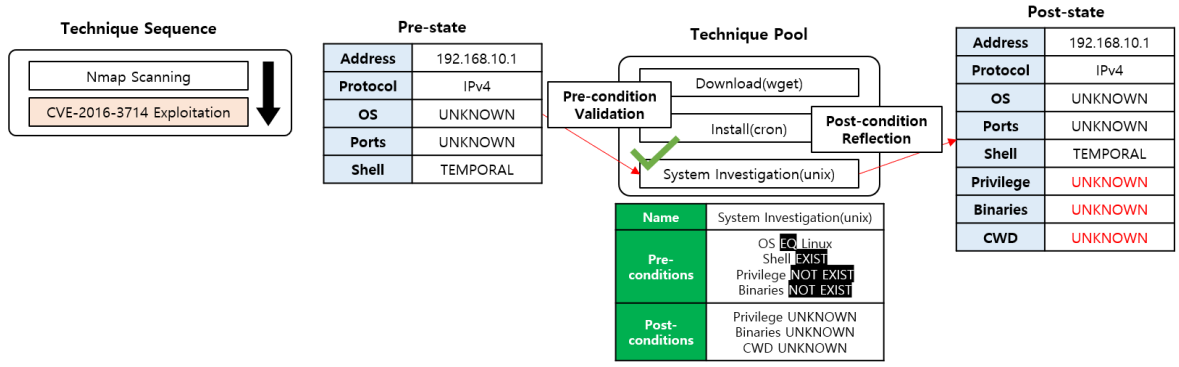

Figure 4. The example of the process of scenario generation: system investigation chosen.

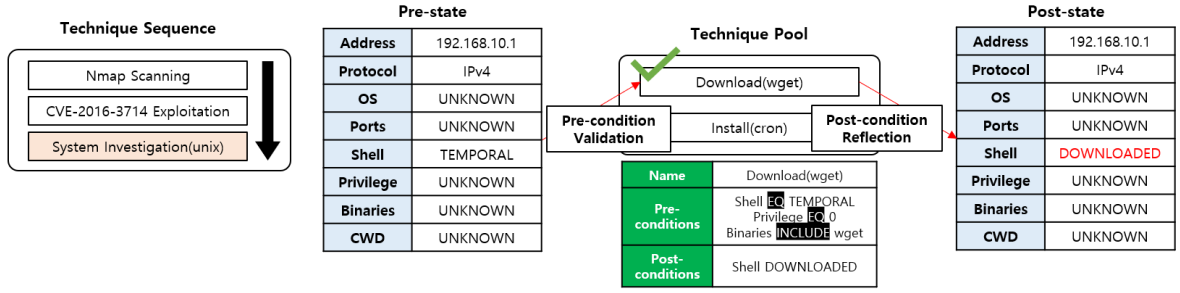

Figure 5. The example of the process of scenario generation: remote access tool (RAT) download chosen.

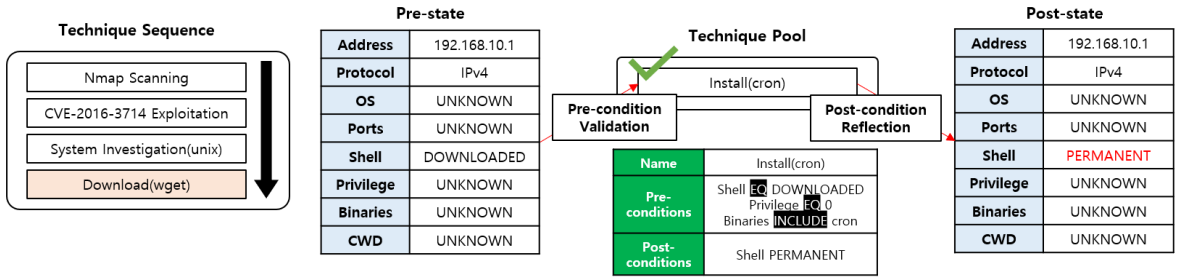

Figure 6. The example of the process of scenario generation: RAT installation chosen.

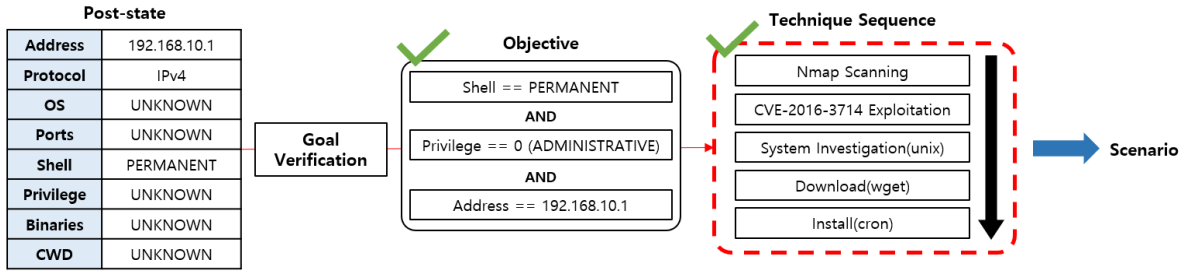

Figure 7. The example of the process of scenario generation: completion of generating a scenario.

Figure 2 describes an initial status of the scenario generation. It contains nothing on the technique sequence. The pre-state stores the address and the protocol information of the target system which were 192.168.10.1 and IPv4, respectively. The red team agent searches for an available attack technique from the technique pool through the pre-condition validation. In the figure, it shows that the 'Nmap scanning' technique was selected because its pre-conditions are satisfied; the target address does exist (Address EXIST), the protocol being used was IPv4 (Protocol EQ IPv4), and the ports were not examined (Ports NOT EXIST). The red team agent selects the Nmap scanning technique and creates the post-state through the post-condition reflection.

The next step is depicted in Figure 3. The technique sequence is no more empty and it contains one attack technique, Nmap scanning. The post-state created in Figure 2 is used as the pre-state in the current step. The scenario generator performs the pre-condition validation to find the next attack technique based on its pre-conditions. The figure shows 'CVE-2016-3714 exploitation' passes the validation. Successively, it performs the post-condition reflection to create the post-state. The similar process is repeated in Figures 4–6 to select attack techniques in sequence.

At last, Figure 7 shows how the goal is achieved as a result of the goal verification. After the goal verification, the process continues if the post-state does not meet the goal of the red team agent. If the goal is achieved, the technique sequence will be exported as a valid attack scenario.

4.5.2. Performing Attack Scenarios

This section provides an example regarding performing the attack scenario, and it follows Algorithm 3. Unlike the scenario generation, attacks are performed using the actual state of the

red team agent. The keyword UNKNOWN is not used in this phase because all unknown information is revealed while the real attacks are performed. In some cases, it is possible some scenarios go out of use, as mentioned in Section 4.3.

The scenario performance process is shown in Figures 8–13. Each figure contains 'scenario,' 'pre-state,' 'post-state' information. 'Scenario' is the sequence of the attack techniques derived by the scenario generator. 'Pre-state' and 'post-state' are the states of the red team agent before and after it uses the attack technique, respectively. 'Pre-condition validation' validates the pre-state of the red team agent satisfies the pre-condition of the attack technique to be used next. Unlike the scenario generation assumed the post-state, 'post-condition reflection' in the scenario performance updates the post-state with the actual state after the attack technique is operated. 'Goal verification' process is similar to it on the scenario generation process. It checks if the goal is achieved every time the post-state is reflected, and if the goal is achieved, the red team agent terminates with the successful attack.

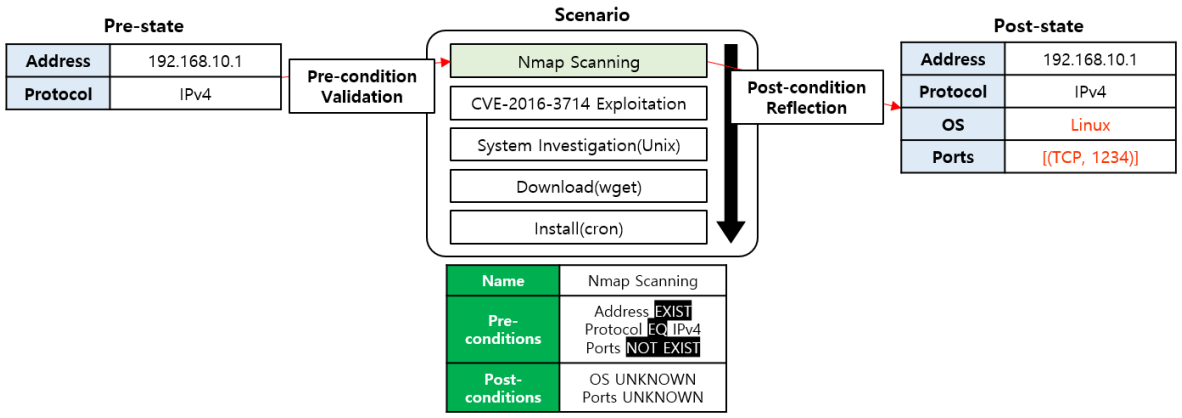

Figure 8. The example of the process of scenario performance: port scanning.

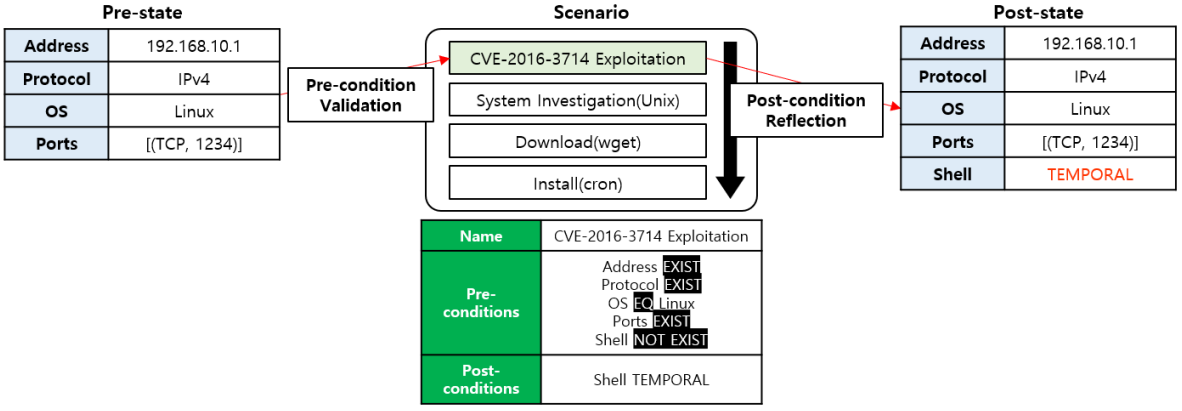

Figure 9. The example of the process of scenario performance: CVE-2016-3714 exploitation.

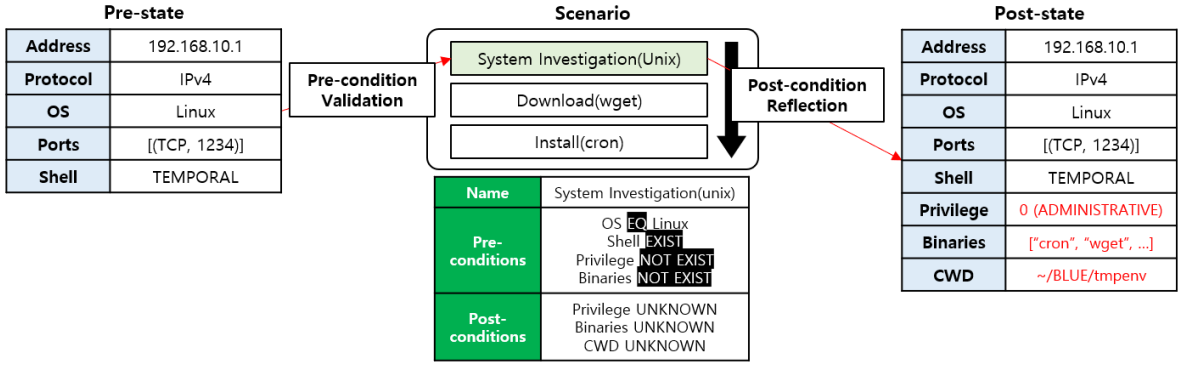

Figure 10. The example of the process of scenario performance: system investigation.

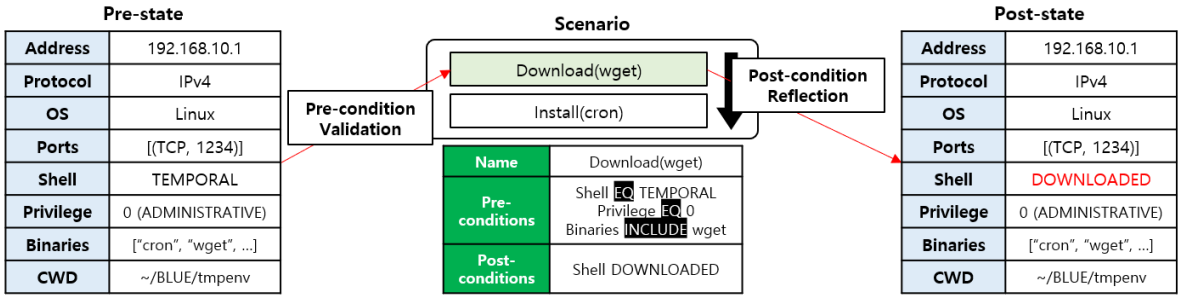

Figure 11. The example of the process of scenario performance: RAT download.

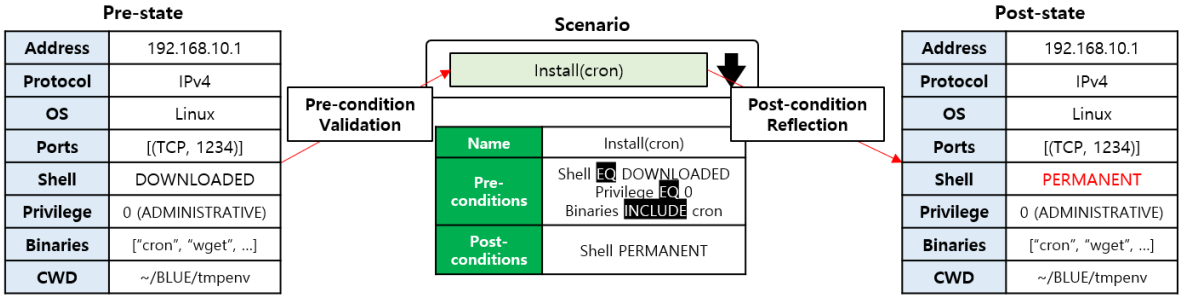

Figure 12. The example of the process of scenario performance: RAT installation.

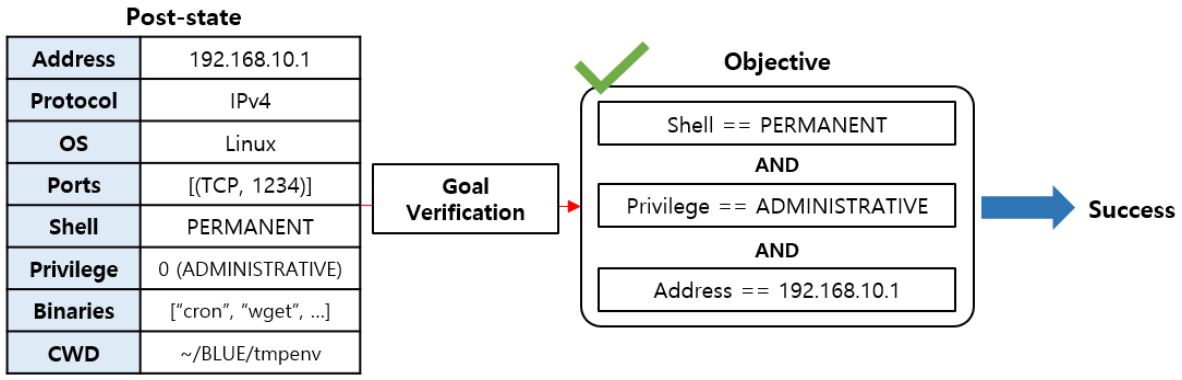

Figure 13. The example of the process of scenario performance: completion of performing a scenario.

Figure 8 shows the example of one scenario performance. The address and protocol information of the target system are stored in the pre-state, 192.168.10.1 and IPv4, respectively. The first attack technique of the scenario is Nmap scanning, and the red team agent manages the pre-condition validation on it. If the result shows the pre-conditions of Nmap scanning matches with the pre-state, the attack technique will be used for the attack. Executing the port scanning (Nmap scanning) could gather the following information of the target system; the operating system is Linux and the opened port is '(TCP, 1234)'. Then, the red team agent makes the pre-state be updated to get the post-state.

Figure 9 describes the next step of Figure 8. The post-state of the previous step becomes the pre-state. The first attack in the remainder of the scenario is CVE-2016-3714 exploitation. The red team agent performs the pre-condition validation on the attack technique to exploit CVE-2016-3714. After the technique is executed, the agent operates the exploitation. If the result delivered is a success, the red team agent is able to gain a temporary shell and updates the pre-state to get the post-state. Figures 10–12 imply the similar processes are repeated.

The result of the scenario is depicted in Figure 13, and it shows the goal of the scenario has been achieved. As follows, the red team agent stops the attack performance judging the scenario leads to the successful attack on the target system. The general case is to achieve the goal after using all attack techniques in the scenario. This is because attack scenarios are generated just as much as the adversary emulation needs.

5. Blue Team Agent

This section describes the overall architecture of the blue team agent and how it carries out cyber-defenses. For automated defenses, we defined 'defense technique.' The 'defense technique' is a unit of defenses for the blue team agent.

It differs much from the red team agent in terms of the architecture and ways of running. The differences between two agents come from their initial conditions, roles (attack or defense) and goals. The techniques used by both agents are different.

For the red team agent, we set up pre-conditions and post-conditions and defined a state to launch appropriate attacks. On the other hand, the execution of the blue team agent is more passive and the conditions for defense techniques are much less compared to the red team agent. Furthermore, the blue team agent's defense is simpler than the red team agent's attack organization.Because attack techniques composing the attack scenario are linked, even defending one of the techniques can cut off the scenario and neutralize the attack. For these reasons, we decided not to devise a data model for the blue team agent.

5.1. Overall Architecture of Blue Team Agent and Blue Team Server

Figure 14 depicts the blue agent server and the overall architecture of the blue team agent. In the figure, the blue team agent is installed on a node (system). It contains defense information and backups. The defense information can include target file list, file hashes, intrusion detection system (IDS) rules, and so on. The blue team agent communicates with the server to download or upload data.

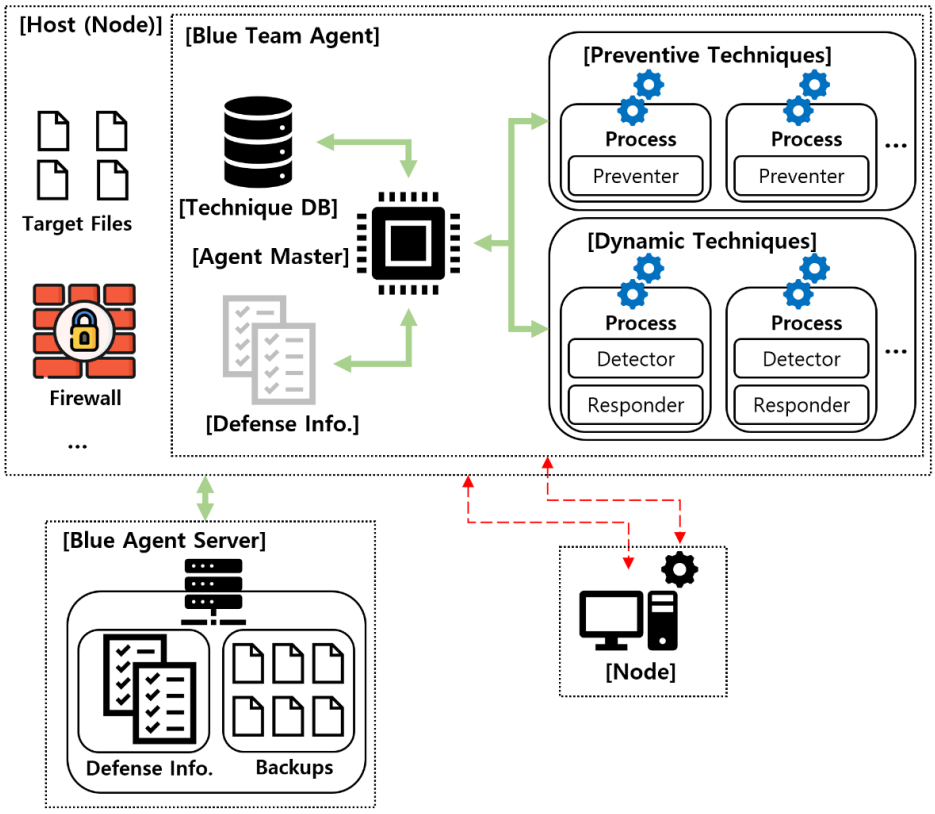

Figure 14. Overall architecture of blue team agent and blue team server.

There are two types of the defense techniques: 'preventive techniques' and 'dynamic techniques.' Preventive techniques can be used to defend the node before detecting attacks. The blue team agent investigates weak points of the system such as wrong authorizations or misconfigurations.If any weak points are found, the agent uses the preventive techniques to mitigate them. Secondly, the dynamic

technique helps the blue team agent detect incoming attacks and confront them. One example is a defense technique to monitor a specific directory for its integrity. While monitoring, if the integrity is broken, the technique makes the blue team agent download a backup of the corrupted file and restore them. After defense techniques are executed, they stay in the system and run as background processes.

5.2. Overall Process and Algorithm

The overall process of the blue team agent is depicted in Algorithm 4. First of all, it calls 'basic_check()' to determine if the system has weak configurations. Then, 'mitigate_weak_files()' checks if running web-servers or databases are misconfigured from the security point of view. If any, it modifies the configuration files to mitigate weaknesses.

After mitigating misconfigurations, it enables the firewall and the IDS with their rules. 'enable_firewall()' adds the firewall rules and enables the firewall to observe network traffic. 'enable_IDS()' creates IDS processes using the IDS rules to keep track of network traffic or local files.

The blue team agent also monitors files to check their integrity. It calculates hash values of files and stores their backups in the blue agent server. Then it calls 'monitor_files()' and how it operated is on Algorithm 5.The function opens a log file that has log records about the file access, and goes into the loop iteration on each line of the log file. If a suspicious case (e.g., the monitored file was accessed and modified illegally) is found, then the blue team agent regards the case as an attack situation and restores the file.

Algorithm 4: The overall process.

```
initialization();
basic_check();
mitigate_weak_files(files and keywords);
enable_firewall();
enable_IDS(ids_rule_file);
foreach file_name, file_path in new_target_files do
    hash(file_name);
    add_to_monitoring(file_name, file_path);
end
while true do
    monitor_files();
    sleep(30);
end
```

Algorithm 5: monitor_files.

```
monitoring_log ← open(monitoring_log_file);
hash_list ← open(file_hashes_being_monitored);
foreach json_line in monitoring_log do
    file_name ← json_line["target_file_path"];
    action ← json_line["action"];
    if file_name in hash_list AND action is "UPDATED" then
        restore_file(file_name);
        clear_log_file();
    end
end
```

6. Experiment and Result

6.1. Experiment Setup

We conducted three scenario-based experiments to verify the feasibility of the red team agent and the blue team agent. Through the experiments, we expect to confirm the red team agent automatically launches attacks, and the blue team agent performs automated defenses as we intended on the scenarios. We created eight attack techniques for the red team agent and two defense techniques for the blue team agent. For the experiments, we prepared a Linux system with two vulnerabilities (CVE-2016-3714 and CVE-2015-5958) and a misconfiguration of *sudoers*. Table 3 describes the two vulnerabilities and one misconfiguration.

Table 3. Vulnerabilities and misconfiguration for experiments.

Name	Description
Vulnerability CVE-2016-3714 [16]	The target system has a daemon owned by root and the daemon uses a vulnerable version of 'ImageMagick' with CVE-2016-3714 through port 1234. Exploiting it, the red team agent performs RCE attack against the system with administrator privilege.
Vulnerability CVE-2015-5958 [17]	The target system has an Nginx web server running a vulnerable version of 'phpFileManager' with CVE-2015-5958 through port 3448. Exploiting it, the red team agent performs RCE attack against the system.
Misconfiguration of *sudoers*	There exists an user 'www-data' in the target system, and it is configured as follows: "www-data ALL=NOPASSWD: /usr/bin/find"; the user www-data can run the command '/user/bin/find' with administrator privilege without a password.

The first experiment was set up to enable only the red team agent.The red team agent generated attack scenarios using the eight attack techniques.We expected two attack scenarios to successfully attack the target on this experiment. One is to exploit CVE-2016-3714 and it is on the left part of Figure 15. The other is to exploit both CVE-2015-5958 and the misconfiguration, as depicted in the left part of Figure 16 In the second experiment, both the red team agent and the blue team agent were activated. Only one defense technique was used in the experiment: the mitigation for CVE-2016-3714. With these settings, the expected result was a failure in the first scenario and a success in the second one. The right side of Figure 15 and the left side of Figure 16 show the expected result on this experiment. The last experiment was set up similar to the second experiment, and there is one more defense technique, mitigating the misconfiguration. Both of the scenarios were expected to be failed with these settings.

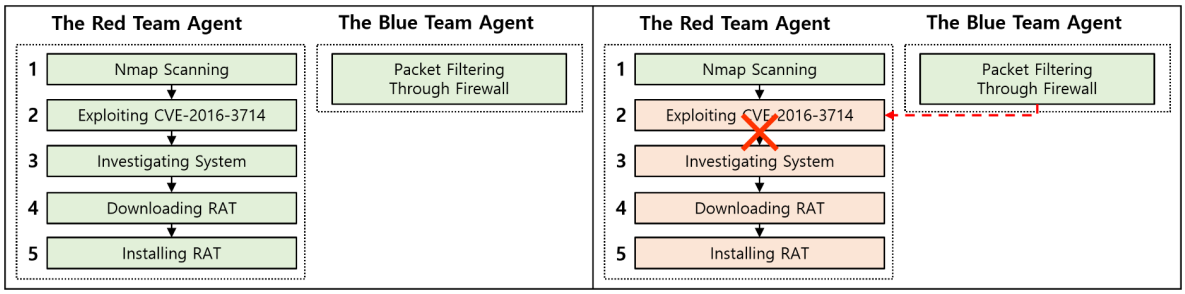

Figure 15. Available techniques for the red team agent and the blue team agent on the system with CVE-2016-3714 (left) and the red team agent's failure of exploiting CVE-2016-3714 by the blue team agent's mitigation (right).

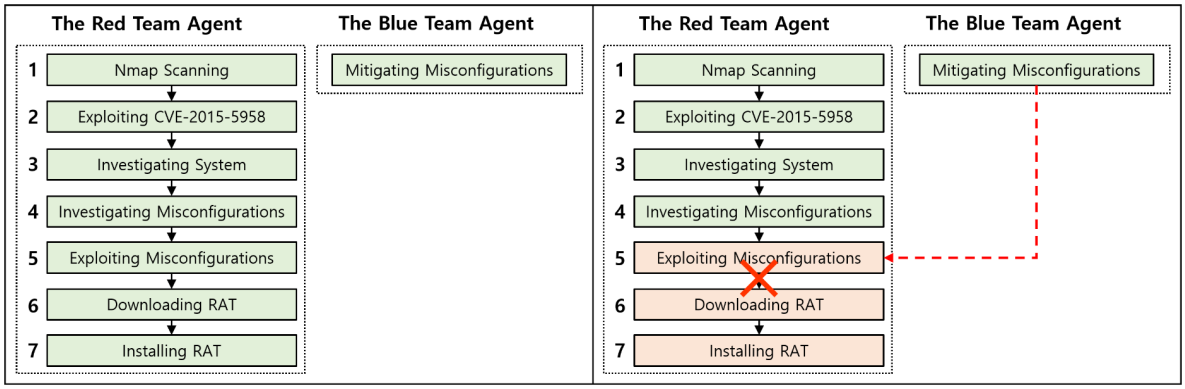

Figure 16. Available techniques for the red team agent and the blue team agent on the system with CVE-2015-5958 and misconfiguration of *sudoers* (left) and the red team agent's failure of exploiting the misconfiguration by the blue team agent's mitigation (right).

6.2. Experiment 1

When the red team agent went into action, it generated several attack scenarios with the eight attack techniques according to the pre-conditions and the post-conditions of each attack technique, and its goal. Figure 17 shows the red agent generated multiple scenarios.The red team agent used them one by one until it achieved its goal. Among the many scenarios, the successful scenario was the one including CVE-2016-3714 exploitation.

```
Attack Scenario #1
Attack Scenario ==========
[0] nmap for searching open ports
[1] CVE-2015-5958 for RCE(unix)
[2] System Investigation(unix)
[3] Download(wget)
[4] Install(cron)
Result : INIT
Last Technique : 0
===============Attack Scenario #2
Attack Scenario ==========
[0] nmap for searching open ports
[1] CVE-2015-5958 for RCE(unix)
[2] System Investigation(unix)
[3] Download(wget)
[4] Misconfiguration_Investigation(sudo)
[5] Install(cron)
Result : INIT
Last Technique : 0
===============Attack Scenario #3
Attack Scenario ==========
[0] nmap for searching open ports
[1] CVE-2015-5958 for RCE(unix)
[2] System Investigation(unix)
[3] Download(wget)
[4] Misconfiguration_Investigation(sudo)
[5] Exploit_Misconfiguration(sudo find)
[6] Install(cron)
Result : INIT
Last Technique : 0
===============Attack Scenario #4
Attack Scenario ==========
[0] nmap for searching open ports
[1] CVE-2015-5958 for RCE(unix)
[2] System Investigation(unix)
[3] Misconfiguration_Investigation(sudo)
[4] Download(wget)
[5] Install(cron)
Result : INIT
Last Technique : 0
===============Attack Scenario #5
Attack Scenario ==========
[0] nmap for searching open ports
[1] CVE-2015-5958 for RCE(unix)
```

Figure 17. Examples of generated scenarios.

The left side of Figure 18 displays the screen of the successful attack scenario. The five attack techniques were used by the red team agent sequentially on the system: 'Nmap scanning', 'exploiting CVE-2016-3714', 'investigating system', 'downloading RAT', and 'installing RAT'. Nmap port scanner found out the port 1234 had been opened. Exploiting CVE-2016-3714, the red team agent was able to perform the RCE attack against the 'ImageMagick' daemon. After it ran the RCE attack properly, the red team agent got a shell of the system with administrative privilege.From this point, it could send the system some commands through the gained shell. The red team agent investigated the inside of the system. At last, the red team agent downloaded and installed an RAT program to make it run periodically with a job scheduler utility.

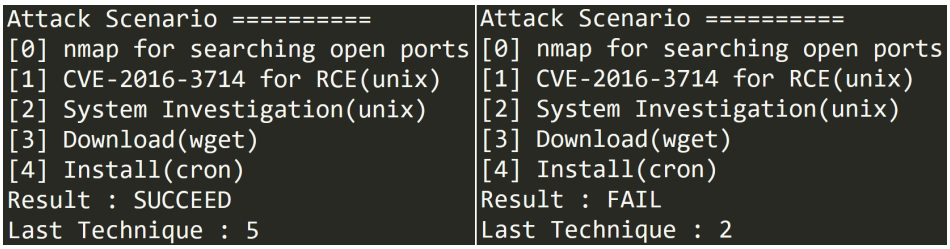

Figure 18. The success (left) and the failure (right) of the red team agent's attacks exploiting CVE-2016-3714 on the system.

6.3. Experiment 2

Unlike the first experiment, the blue team agent operated one defense technique, the mitigation for CVE-2016-3714 exploitation. To use the technique, it communicated with the blue agent server to get firewall rules to apply on the firewall and enable the firewall. In addition, the rules include a packet filtering rule to drop the payload of exploiting CVE-2016-3714.

After the blue team agent used the defense technique, the red team agent failed to use every attack scenario which includes CVE-2016-3714 exploitation. The right side of Figure 18 shows the result of performing the failed scenario.The red team agent tried to exploit CVE-2016-3714 after discovering the active port number (i.e., 1234) of the daemon with the vulnerability.However, the exploitation failed because of the rules of the firewall, the defense technique of the blue team agent. The failure of CVE-2016-3714 exploitation led to no satisfaction of the related post-conditions, therefore the pre-conditions of the upcoming attack technique, 'investigating system', were not satisfied. By such results, the attack scenario failed.

Instead of the failed scenario, we expected the other scenario with CVE-2015-5958 exploitation and misconfiguration exploitation to succeed. As expected, the left side of Figure 19 shows the scenario successfully attacked the target. The seven attack techniques were used by the red team agent sequentially on the system: 'Nmap scanning', 'exploiting CVE-2015-5958', 'investigating system', 'investigating misconfigurations', 'exploiting misconfigurations', 'downloading RAT', and 'installing RAT'. Nmap port scanner found out the port 3448 had been opened. Exploiting CVE-2015-5958, the red team agent was able to perform the RCE attack against the 'phpFileManager'. After it ran the RCE attack properly, the red team agent got a shell of the system. Through the shell, it could send the system commands to investigate the system and to examine misconfigurations. After finding the misconfiguration, the red team agent exploited it to obtain the administrative privilege. With the escalated privilege, the red team agent downloaded and installed an RAT program to make it run periodically with a job scheduler utility.

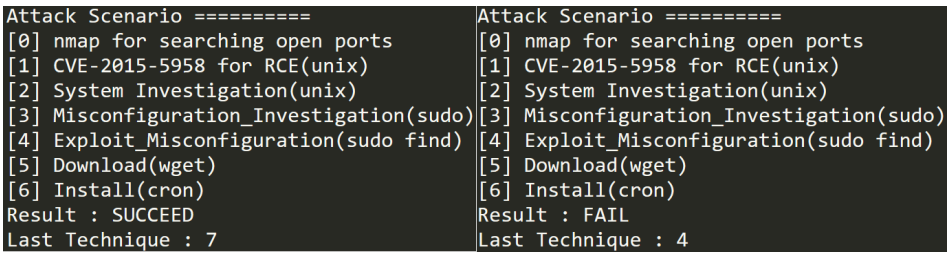

Figure 19. The success (left) and the failure (right) of the red team agent's attacks exploiting CVE-2015-5958 and misconfiguration of *sudoers* on the system.

6.4. Experiment 3

Similarly to the second experiment, the red team agent and the blue team agent were all active. It added one more defense technique to the settings of the second experiment, mitigation

for misconfiguration. The added defense technique examines '/etc/sudoers' file and removes any misconfigurations on the file.

Using two defense techniques, the red team agent resulted in failure every scenario with CVE-2016-3714 or misconfiguration exploitation.How the attack scenario using CVE-2016-3714 exploitation failed is described on Section 6.3. The failure of the scenario exploiting CVE-2015-5958 and misconfiguration is on the right side of Figure 19. The red team agent used the attack techniques from 'Nmap scanning' to 'system investigation' without any failure. However, 'misconfiguration investigation' did not succeed because the blue team agent already removed the misconfiguration. Accordingly, one pre-condition of the next attack technique, 'exploit_misconfiguration(sudo find)' was not satisfied. This scenario resulted in the failure and all the remaining scenarios failed as well. Then, the red team agent finished the attack without an achievement. Figure 20 shows the screen of all scenarios failed and the red team agent stopped its attacks.

```
Attack Scenario ==========
[0] nmap for searching open ports
[1] CVE-2015-5958 for RCE(unix)
[2] System Investigation(unix)
[3] Misconfiguration_Investigation(sudo)
[4] Download(wget)
[5] Exploit_Misconfiguration(sudo find)
[6] Install(cron)
Result : FAIL
Last Technique : 2
===============
Attack Scenario ==========
[0] nmap for searching open ports
[1] CVE-2015-5958 for RCE(unix)
[2] System Investigation(unix)
[3] Misconfiguration_Investigation(sudo)
[4] Exploit_Misconfiguration(sudo find)
[5] Download(wget)
[6] Install(cron)
Result : FAIL
Last Technique : 2
===============
Attack Scenario ==========
[0] nmap for searching open ports
[1] CVE-2016-3714 for RCE(unix)
[2] System Investigation(unix)
[3] Download(wget)
[4] Install(cron)
Result : FAIL
Last Technique : 2
=========================
[Listener] Start stopping server
[Listener] Server stopped
```

Figure 20. An example of the failure of all scenarios.

6.5. Dataset and Source Code

We released the dataset and source codes used in our experiments to foster further research. We make our dataset and source code available at GitHub [18]. This repository includes the implemented agents, 266 attack techniques and 167 defense techniques.

7. Discussion

In this paper, we conducted experiments based on the adversarial emulation. We tested several scenarios by changing the vulnerabilities to be exploited, also by turning on and off the blue team agent's defense functionality. All of the demonstrated attack techniques and vulnerabilities are being used by hackers in the real world. In these scenarios, we showed the red team agent could successfully penetrate into the target system when the blue team agent was disabled. We showed the red team agent failed to gain the target system's privilege when the blue team agent was enabled. To summarize, the results of the experiments confirm the proposed emulation operates well as we designed.

Table 4 shows the comparison of our proposed agents and other adversary emulation tools we surveyed.

Table 4. Comparison among adversary emulation tools.

	# of Implementations (name)	Defense Coverage	Available Platforms
Our Agents (Red Team Agent +Blue Team Agent)	8 (attack techniques) +2 (defense techniques)	PDR	Linux
MITRE CALDERA+CASCADE	91 (abilities)	D	Windows, Linux, macOS
Metta	57 (actions)	-	Windows, Linux, macOS
Atomic Red Team	619 (atomic tests)	-	Windows, Linux, macOS
APTSimulator	26 (test-sets)	-	Windows

The strength we can put forward is that our emulation framework covers the automated defenses. To this end, we design and implement the blue team agent that supports prevention, detection, and response to attacks. Most of adversary emulation tools do not support defense action. Only the MITRE CASCADE covers attack detection but does not cover the whole process of the defense (prevention, detection and response).

Limitation

We investigated various techniques for both red teams and blue teams; and we implemented nine techniques as a proof of concept. Although we demonstrated the suggested techniques worked well, we did not implement all known red team and blue team techniques in this experiment. For this reason, we will implement and test more techniques as future work. There is one more thing to improve the agents further. We designed the red team agent to attack adjacent systems during the attack process. To emulate the real world's network configuration more, we need to consider a more complex network structure that has multi-network interfaces and dynamic access-control functionality. To solve this issue, we will improve the network reconnaissance functions for the red team agent to bypass the network-based access control. We will implement a backdoor agent that can relay connections between nodes to emulate reverse connections after installing a backdoor program.

8. Conclusions

In this paper, we proposed an adversary emulation model to support an efficient adversary emulation. We implemented an adversary emulation framework composed of the red team agent and the blue team agent. Based on the carefully designed scenario, our evaluation results show that our model effectively performs the automated attacks and defenses. We showed our model's merits by comparing with other adversary emulation tools. Although we currently have limitations because we tested a few scenarios, we believe that the proposed model can be an efficient tool to find out the possible attack paths. Our model will contribute to reducing a lot of overhead coming from the manual test. Our model will be helpful to security administrators to assess their organization's system and network routinely.

In the future, we will develop more attack techniques and defense techniques. The more diverse scenarios can be covered, the more efficient adversary emulation will be achieved.

Author Contributions: Conceptualization, methodology and software, J.D.Y. and G.L.; validation, D.K. and S.S.; writing—original draft preparation, J.D.Y.; writing—review and editing, J.D.Y., E.P. and H.K.K.; supervision, H.K.K.; project administration, M.K.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Agency for Defense Development grant number UD190002ED.

Acknowledgments: This work was supported by the Agency for Defense Development under the contract UD190002ED.

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

References

1. 2018: A Year of Cyber Attacks. Available online: https://www.hackmageddon.com/2019/01/15/2018-a-year-of-cyber-attacks (accessed on 18 March 2020).
2. Browse vUlnerabilities by Date. Available online: https://www.cvedetails.com/browse-by-date.php (accessed on 18 March 2020).
3. Adversary Simulation Becomes a Thing. Available online: https://blog.cobaltstrike.com/2014/11/12/adversary-simulation-becomes-a-thing (accessed on 18 March 2020).
4. Caldera. Available online: https://www.mitre.org/research/technology-transfer/open-source-software/caldera (accessed on 18 March 2020).
5. Cascade GitHub. Available online: https://github.com/mitre/cascade-server (accessed on 18 March 2020).
6. Metta GitHub. Available online: https://github.com/uber-common/metta (accessed on 18 March 2020).
7. Atomic Red Team. Available online: https://atomicredteam.io (accessed on 18 March 2020).
8. APT Simulator GitHub. Available online: https://github.com/NextronSystems/APTSimulator (accessed on 18 March 2020).
9. MITRE ATT&CK. Available online: https://attack.mitre.org (accessed on 18 March 2020).
10. Applebaum, A.; Miller, D.; Strom, B.; Korban, C.; Wolf, R. Intelligent, automated red team emulation. In Proceedings of the 32nd Annual Conference on Computer Security Applications, Los Angeles, CA, USA, 5–9 December 2016; pp. 363–373.
11. Miller, D.; Alford, R.; Applebaum, A.; Foster, H.; Little, C.; Strom, B. *Automated Adversary Emulation: A Case for Planning and Acting with Unknowns*; MITRE: McLean, VA, USA, 2018.
12. Kavak, H.; Padilla, J.J.; Vernon-Bido, D. A characterization of cybersecurity simulation scenarios. In Proceedings of the 2016 Spring Simulation Conference (CNS), Pasadena, CA, USA, 3–6 April 2016.
13. Kuhl, M.E.; Sudit, M.; Kistner, J.; Costantini, K. Cyber attack modeling and simulation for network security analysis. In Proceedings of the 2007 Winter Simulation Conference, Washington, DC, USA, 9–12 December 2007; pp. 1180–1188.
14. Applebaum, A.; Miller, D.; Strom, B.; Foster, H.; Thomas, C. Analysis of automated adversary emulation techniques. In Proceedings of the Summer Simulation Multi-Conference, Bellevue, WA, USA, 9–12 July 2017; pp. 1–12.
15. Fikes, R.E.; Nilsson, N.J. STRIPS: A new approach to the application of theorem proving to problem solving. *J. Artif. Intell.* **1971**, *2*, 189–208.
16. CVE-2016-3714. Available online: https://cve.mitre.org/cgi-bin/cvename.cgi?name=2016-3714 (accessed on 18 March 2020).
17. CVE-2015-5958. Available online: https://cve.mitre.org/cgi-bin/cvename.cgi?name=2015-5958 (accessed on 18 March 2020).
18. Red-Blue-Agents GitHub. Available online: https://github.com/hksecurity/Red-Blue-Agents (accessed on 18 March 2020).