

WiP: Steerability of Autonomous Cyber-Defense Agents by Meta-Attackers

Luis Burbano
lburbano@ucsc.edu
University of California Santa Cruz
Santa Cruz, California, USA

Hampei Sasahara
sasahara@sc.eng.isct.ac.jp
Institute of Science Tokyo
Tokyo, Japan

Alvaro Cardenas
alacarde@ucsc.edu
University of California Santa Cruz
Santa Cruz, California

Abstract

AI agents are increasingly automating several traditional manual tasks. One area where AI agents show promise is computer incident response, as this is a significantly slow and sometimes tedious process managed by operators who are overwhelmed with alarms. However, before deploying these AI systems, we need to make sure that an attacker cannot exploit them. This paper formally analyzes an attacker who has partially compromised an autonomous cyber defense agent. Our goal is to understand how such an attacker can steer the defenses and manipulate the system to achieve its objectives. Our results can help defenders identify the most critical components of their defenses and harden resources that (if compromised) may give an attacker a large advantage.

ACM Reference Format:

Luis Burbano, Hampei Sasahara, and Alvaro Cardenas. 2025. WiP: Steerability of Autonomous Cyber-Defense Agents by Meta-Attackers. In *Proceedings of Hot Topics in the Science of Security Symposium (HotSoS) (HotSoS '25)*. ACM, New York, NY, USA, 9 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 Introduction

As computer networks continue to advance technologically, they provide new functionalities that increase their attack surface and allow sophisticated threats to move across the network. Detecting and removing these attackers from partially compromised networks before they reach their targets is a critical aspect of cyber resilience. The state of the art security practices to respond to attacks involve a delay where human analysts need to review the indicators of compromise in the network and then follow incident response playbooks. Unfortunately, an increasing number of devices on these networks and a large number of false alarms prevent human analysts from handling all alerts. Furthermore, human-based recovery is often a lengthy process, and current incident response practices lag behind the attacker's ability to find new network vulnerabilities and keep their presence in the network, allowing the attacker to remain in the network.

To address these limitations, researchers are developing and testing new AI agents to respond to attacks autonomously [15]. These new Autonomous Cyber Defense (ACD) agents provide real-time

responses to attacks [11, 12]. ACD agents can automatically block or quarantine malicious traffic, isolate infected endpoints, reset user or device credentials, and reboot or reimage devices in the core network. The goal is that this proactive recovery will (1) prevent an adversary from retaining a foothold in the network and (2) minimize the time between detection and removal of the attacker from the network.

These agents, however, will be deployed on a network with adversaries, and parts of the infrastructure on which the agents rely may be compromised. An attacker can potentially (1) alter the information that the ACD reads to make decisions (we call these sources of information "sensors"), or (2) alter the actions the ACD takes (we call tampering with the actions of the ACD "actuation" attacks). For example, an attacker may compromise a machine and send fake logs to the ACD, implying that it is being targeted by another device, and after receiving this information, the ACD agent might decide to block or quarantine the framed device.

This paper focuses on the risks and consequences of test-time attacks on ACDs. We create a formal model of the problem and evaluate how a partially compromised ACD can be manipulated by attackers at test time. Our model and results can help the defenders identify the most critical components of the ACD agents; for example, if the ACD is more steerable when the attacker compromises "sensor 1" than "sensor 2", then the defender should allocate more resources to harden "sensor 1."

Our contributions include:

- We define a new threat model that we call *meta-attacker* and formulate the *steerability* problem in an ACD infrastructure.
- We propose two new algorithms to find the impact of different types of attacks on the ACD infrastructure.
- We design an experiment based on the CAGE Challenge 2 [1], a popular competition for testing ACD agents. Our experiments show how our analysis can find new insights into this problem, such as the fact that the optimality of meta-attacks depends on the time deadline the attacker has to reach its target in the network.

The remainder of the paper is organized as follows. Section 2 gives a brief introduction to ACD. Section 3 formulates the problem of ACD with a meta-attacker and introduces the mathematical framework. Section 4 presents tools to analyze the steerability problem from the attacker's point of view. Section 5 presents a solution to the steerability of a network with a meta-attacker. Sections 6 and 7 show a study case and the results. Section 8 concludes the paper.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
HotSoS '25, April 1–3, 2025.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
<https://doi.org/XXXXXXX.XXXXXXX>

2 Background

ACD agents have primarily been designed by the use of Reinforcement Learning (RL). Ridley et al. [12] implemented one of the first efforts by using Q-learning to obtain agents that respond to threats. Since then, several efforts have addressed this problem using deep RL (DRL) in hopes that deep neural networks can learn a better policy.

DRL approaches to train ACD agents include deep Q-learning [2] and Proximal Policy Optimization (PPO) [7]. A key aspect of the performance of these agents is the design of the reward function [4]. To avoid overfitting the RL policy to a given attacker, curriculum learning [5] has been proposed as a way to train RL ACD agents that generalize to different settings.

Most of the previous work focuses on designing new ACD agents without discussing attacks on this autonomous defense. This is important because ACD agents operate in an adversarial environment, where advanced attackers can not only target the network but also the ACD agents [?]. Automated responses might be a double-edged sword in that they can be very good at protecting a network, but at the same time, they might be abused by attackers.

An attacker can send false information to the ACD agent in the hope that the ACD agent takes an action that the attacker wants. Similarly, an attacker can block or change the intended actions of the ACD agent before reaching their target. We call an attack that compromises the integrity of the ACD infrastructure a **meta-attack** because the adversary exploits an agent that defends the network from “classical” attacks. Meta-attacks are a new risk for ACD agents, and therefore, we need to develop new tools to understand the consequences of these attacks and allocate defense resources. In this paper, we make the first attempt to achieve this goal.

3 Problem formulation

Our goal is to propose a formal framework to analyze meta-attacks against ACDs. A common approach to formalize ACD agents using DRL is using probabilistic models, such as Markov Decision Process (MDP) [2, 12, 17] or partially observable MDP (POMDP) [9, 12]. With these formalisms, we propose a framework to analyze the impact of a partially compromised ACD infrastructure.

3.1 Mathematical framework

We consider a network with $n_H \in \mathbb{N}$ hosts, an attacker, and an ACD as in Fig. 1. We model the interaction between the attacker and the defender as two agents that modify a system by their actions. As a consequence of the agent’s action, the network changes from an initial state (e.g., no host is compromised) to a new state (e.g., one host is compromised).

The attacker and the defender share the same network, whose dynamics can be modeled as a POMDP:

$$\tilde{\mathcal{M}} = \langle \mathcal{S}, \tilde{T}, \mathcal{A}^{(d)}, \mathcal{A}^{(a)}, \mathcal{O}^{(d)}, \mathcal{O}^{(a)}, \mathcal{O}^{(d)}, \mathcal{O}^{(a)}, AP, L \rangle, \quad (1)$$

Each element of the POMDP is:

- **State set \mathcal{S} :** The finite state set models the current state of the network and contains information on each host.

- **Action sets:** The finite action sets define the possible actions that the defender $\mathcal{A}^{(d)}$, and attacker $\mathcal{A}^{(a)}$ can take.
- **Transition function:** The transition function $\tilde{T} : \mathcal{S} \times \mathcal{S} \times \mathcal{A}^{(a)} \times \mathcal{A}^{(d)} \rightarrow [0, 1]$ models the probabilistic behavior of a network. Given that the network is in a state $s \in \mathcal{S}$, and that the attacker and defender take actions $a^{(a)} \in \mathcal{A}^{(a)}$, $a^{(d)} \in \mathcal{A}^{(d)}$, the function $\tilde{T}(s'|s, a^{(a)}, a^{(d)})$ describes the probability that the network reaches a new state s' .
- **Observation sets:** the observation sets model the information that the attacker $\mathcal{O}^{(a)}$ and the defender $\mathcal{O}^{(d)}$ obtain from the network after their actions.
- **Observation functions:** They are functions $O^{(i)} : \mathcal{O}^{(i)} \times \mathcal{S} \times \mathcal{A}^{(a)} \times \mathcal{A}^{(d)} \rightarrow [0, 1]$, for $i \in \{d, a\}$ that model the probability that the defender and the attacker obtain the observation $o^{(i)}$, given the state $s \in \mathcal{S}$, and actions $a^{(i)} \in \mathcal{A}^{(i)}$.
- **Set of atomic propositions:** AP is a finite set of atomic propositions, which are labels that provide information about the network at a given state.
- **Labeling function:** this function $L : \mathcal{S} \rightarrow 2^{AP}$ determines the atomic propositions that are true at a state s .

Modeling attacker’s objective: The attacker wants to compromise a subset of hosts. As the state models which hosts are under control, **the attacker’s objective is to steer the POMDP to any state $s \in B \subseteq \mathcal{S}$** , where B is the target set that contains all possible target states (e.g., the crown jewel of the network, such as the designs of the next fighter plane).

Modeling the agents behavior: The defender and the attacker use simultaneously a policy $\pi^{(i)}$, for $i \in \{a, d\}$. The policy assigns the probability of using $a^{(i)} \in \mathcal{A}^{(i)}$ given the observation $o^{(i)} \in \mathcal{O}^{(i)}$. The policies may depend on previous states and previous actions by the attacker and the defender [3].

Networks as a POMDP: The POMDP state $s \in \mathcal{S}$ encodes information about the network, such as the hosts that the attacker has compromised. At each time instant, the attacker and the defender can modify the POMDP state using an action from the set of actions $\mathcal{A}^{(d)}$ and $\mathcal{A}^{(a)}$, respectively. The POMDP transitions from an initial state $s \in \mathcal{S}$ to a new state $s' \in \mathcal{S}$ in a probabilistic way (given by the transition function \tilde{T}). Simultaneously, each agent receives information regarding the state due to their actions, with the functions $\mathcal{O}^{(d)}$ and $\mathcal{O}^{(a)}$. In the next step, the attacker and the defender use the observation and their policies to take a new action and repeat the process.

3.2 Baseline scenario

We first consider the baseline scenario illustrated in Fig. 1. This scenario has a **vanilla attacker** that spawns somewhere in the network—for example, due to a spear-phishing attack. This attacker moves laterally to a different host in the network, where it attempts to obtain administrator privileges. Meanwhile, a defender (an ACD agent) wants to remove any attacker from the network while keeping services available.

Remark: While the vanilla attacker may be aware of the ACD agent and change their strategy to evade the ACD agent, the vanilla attacker cannot affect the integrity of the ACD agent. This is the classical setting of most of the literature on AI-based ACD agents [2, 4, 8, 13–16].

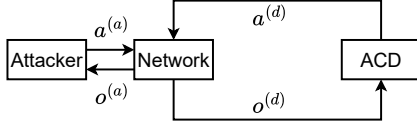


Figure 1: ACD protecting a network against a vanilla attacker.

Our first step is to study the capacity of vanilla attackers to compromise the network (how far in the network they can go, given that the network has a defensive agent) and the time the attacker requires. We can model the vanilla scenario in the POMDP framework as finding the likelihood that the POMDP state eventually reaches a state in the target set B .

PROBLEM 1 (VANILLA SCENARIO – MATHEMATICAL FORMULATION). Consider a network with an ACD and a vanilla attacker modeled with POMDP Eq. (1) and a target set $B \subseteq \mathcal{S}$. We want to 1) determine if the vanilla attacker can steer the POMDP to a state in the target set B from an initial state $s_0 \in \mathcal{S}$, and 2) the time the attacker needs to achieve this objective.

3.3 Meta Attacker scenario

Next, we turn our attention to the case where the attacker compromises the integrity of ACD agents. In particular, we define two attacks: **sensor attacks** (giving wrong information to the ACD agent) and **actuator attacks** (compromising the defensive ACD action). This terminology is aligned with previous work in ACD [18]; where *sensors* refer to devices that obtain information about the state of the network and send it to ACD agents, and *actuators* are the devices that execute the ACD defensive action.

The actuator and sensor meta-attacks can be seen in Fig. 2. The actuator attack compromises the integrity of some actions $a^{(a)}$ by sending to the network $a^{(ma)} \neq a^{(d)}$. The sensor attack compromises some observations so that the ACD sees $o^{(ma)}$, a vector different from the ground truth $o^{(d)}$.

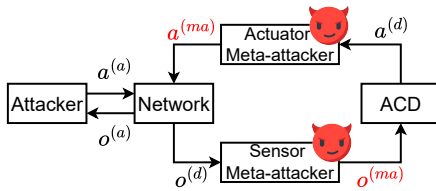


Figure 2: ACD protecting a network against a vanilla attacker and a meta-attacker.

The **goal** of the meta-attacker is to help the vanilla attacker get to its target. In practice, both vanilla and meta-attackers will be launched by the same adversary, but differentiating them allows us to model the problem in a clearer format.

Remark: While the vanilla attacker plays the game within the confines of the POMDP, the meta-attacker can change the structure of the POMDP: it can add or remove transitions between states.

PROBLEM 2 (META-ATTACKER SCENARIO – MATHEMATICAL FORMULATION). Consider the scenario from Problem 1. Given that the

POMDP begins at a state $s_0 \in \mathcal{S}$, we want to find the i) action $a^{(d)}$ or ii) observation $o^{(d)}$ the meta-attacker should compromise to maximize the probability of steering the POMDP to a state in the target set B .

In both cases, the meta-attacker needs to be strategic. First, compromising several actions may create several alerts in the network, making them more detectable. Second, compromising several resources can be complicated. Therefore, we need a methodology to identify the optimal strategy the meta-attacker should follow to maximize the probability of steering the state to the target set.

We now need to look at the theoretical background that can help us solve these problems. In the next Section, we present algorithms to study the reachability problem in MDPs. Later, we will show how to apply those techniques to our POMDP model in Eq. (1).

4 Preliminaries

4.1 MDP as a transition system

Let us assume an MDP as a tuple,

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, AP, L \rangle \quad (2)$$

where $T : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, and $\mathcal{S}, \mathcal{A}, AP, L$ are defined similar to Eq. (1). One approach for solving the reachability problem in MDPs is using transition systems, as it allows us to use graph theory to analyze the MDP.

Directed graphs: We associate a directed graph with the MDP in Eq. (2); we refer to the graph as the underlying MDP graph. An MDP has an underlying graph, given by the tuple $G = (\mathcal{V}, \mathcal{E})$. Each node in the graph represents a state in the MDP, meaning that $\mathcal{V} = \mathcal{S}$. $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges that models the transition from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ given the actions $a \in \mathcal{A}$; each edge is augmented with the probability $\tilde{T}(s'|s, a)$.

4.2 Attacker objective formal definition

We introduce a formal language to formulate the reachability problem mathematically and specify the attacker's objective. Linear temporal logic (LTL) extends propositional logic by adding time operators. We define an LTL formula as,

$$\psi ::= \top \mid a \mid \neg \psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 U \psi_2$$

where \top, \neg, \wedge are true, negation and disjunction from propositional logic. ψ, ψ_1 and ψ_2 are LTL formulas, and $a \in AP$ is an atomic proposition. The operator $\psi_1 U \psi_2$ states that ψ_1 is true at least until ψ_2 becomes true. We define the operator $F\psi = \top U \psi$, which states that ψ becomes true eventually. Finally, we use the constrained eventually operator $F^{\leq n}\psi$ to indicate that ψ becomes true in the next n time steps.

Usually, we define the satisfaction operator \models of LTL formulas using Boolean semantics [6]. In this paper, we use the satisfaction operator \models to express the state properties of the MDP graph. We can write the condition that the state s eventually arrives at some state in B as $s \models FB$.

4.3 Reachability in Markov Decision Process (MDP)

With the formal grammar, we can now proceed to formulate the reachability problem in MDP mathematically. We compute the maximum probability the MDP arrives at a state in B ,

$$Pr_{\mathcal{M}}^{max}(s \models FB) = \sup_{\pi} Pr_{\mathcal{M}}^{\pi}(s \models FB),$$

where the supremum searches over all policies $\pi^{(a)}$. Let us define the vector $(x_s)_{s \in \mathcal{S}}$, with $x_s = Pr_{\mathcal{M}}^{max}(s \models FB)$. We want to compute the values of x_s for all $s \in \mathcal{S}$.

The solution to this problem is presented in [3, Theorem 10.105]. It is an algorithm to obtain the probability of steering the MDP to a state in the target set after infinite time. However, we also study the **constrained time** steerability problem.

We use an iterative algorithm, *value iteration*, for finding the maximum probability the attacker can steer the MDP to a target state in B after n steps. We denote the probability as,

$$x_s^{(n)} = Pr_{\mathcal{M}}^{max}(s \models F^{\leq n}B). \quad (3)$$

Let us define the set $Pre^*(s)$ as the states $s' \in \mathcal{S}$ that can eventually reach s (we omit the formal definition for space constraints). That is, there is a path from s' to s in the underlying MDP graph. We also define $Pre^*(B) = \cup_{s \in B} Pre^*(s)$.

Value iteration [3]: For the states $s \in Pre^*(B) \setminus B$, $x_s = \lim_{n \rightarrow \infty} x_s^{(n)}$, where,

$$x_s^{(0)} = 0 \text{ and } x_s^{(n+1)} = \max \left\{ \sum_{s' \in \mathcal{S}} T(s|s', a) x_{s'}^{(n)} \mid a \in \mathcal{A} \right\}$$

and, $x_s^{(n)} = 1$ if $s \in B$ and $x_s^{(n)} = 0$ if $s \notin Pre^*(B)$.

With this algorithm, we can study the probability the attacker steers the MDP from an initial state to another state in the target set after a constrained amount of time.

5 New Algorithms to Quantify ACD Steerability

The algorithm we presented to analyze the reachability is valid for MDPs. However, we model the network as POMDP with two agents, where those algorithms do not apply directly. In this Section, we show how we can use the reachability algorithms to analyze networks. Then, we encode the reachability with a meta-attacker as optimization problems and solve them.

5.1 Reachability in the baseline network

We consider an ACD agent that already has a fixed policy $\pi^{(d)}$. We then model the network as a POMDP with only one agent (the attacker), who has the action set $\mathcal{A} = \mathcal{A}^{(a)}$.

We assume that the defender can observe the full state for notation and presentation simplicity. However, whether the defender can fully observe the state or not does not change our analysis; therefore, the transition probability becomes,

$$T(s'|s, a^{(a)}) = \sum_{d \in \mathcal{A}^{(d)}} \tilde{T}(s'|s, a^{(a)}, d) \pi^{(d)}(d|s) \quad (4)$$

for any $s, s' \in \mathcal{S}$, and $a^{(a)} \in \mathcal{A}^{(a)}$. Consequently, we can use the theory from MDPs that we presented in Section 4. This means

that we have algorithms to study the reachability of networks. We can get the probability that the attacker compromises a host after infinite and constrained time, solving Problem 1.

Although we have solved the reachability problem in the *vanilla* case, we still need tools to study reachability in a network with a meta-attacker.

5.2 Reachability with a meta-attacker

We now formally define the sensor and actuator meta-attackers and the optimization problem they want to solve.

Actuator meta-attacker: Before defining the actuator meta-attacker, let us first introduce some notation. This meta-attacker changes from one defender action $a^{(d)} \in \mathcal{A}^{(d)}$ to a new action $a^{(ma)} \in \mathcal{A}^{(d)}$ when the POMDP is in $s \in \mathcal{S}$. We model the meta-attacker by changing the defender's policy:

$$\bar{\pi}_{s, a^{(d)}, a^{(ma)}}^{(d)}(a|s') = \begin{cases} 0 & \text{if } a = a^{(d)} \\ & \text{and } s' = s \\ \pi^{(d)}(a|s') & \text{if } a = a^{(ma)} \\ + \pi^{(d)}(a^{(d)}|s') & \text{and } s' = s \\ \pi^{(d)}(a|s') & \text{otherwise.} \end{cases} \quad (5)$$

The modification to the defender policy means that the probability of choosing the action $a^{(d)}$ when the MDP is in state s , is zero due to the meta-attacker action. The probability of using the action $a^{(ma)}$ at state s becomes equal to the defender's original policy probability $\pi(a|s)$ plus the probability of the action the meta-attacker compromised $\pi(a^{(d)}|s)$.

We remark that the defender still uses the policy $\pi^{(d)}$. However, the MDP in Eq.(1) will receive the action $a^{(d)}$ sampled from the policy $\bar{\pi}_{s, a^{(d)}, a^{(ma)}}^{(d)}$. Although there are several alternatives to model the actuator meta-attacker behavior, as defining a new MDP and several of its elements, the modeling we present here is enough for our analysis.

The modification of the defender's policy produces a new MDP with a new transition probability, given by Eq. (4) but with the modified policy from Eq. (5). Let us denote as

$$\mathcal{M}(\bar{\pi}_{s, a^{(d)}, a^{(ma)}}^{(d)}) \quad (6)$$

the MDP that models the network when the meta-attacker compromises the action $a^{(d)} \in \mathcal{A}^{(d)}$ at state $s \in \mathcal{S}$, and changes it to $a^{(ma)}$, making the defender's policy as Eq. (5).

As in Eq.(3), we denote $Pr_{\mathcal{M}(\bar{\pi}_{s, a^{(d)}, a^{(ma)}}^{(d)})}^{max}(s_0 \models F^{\leq n}B)$ as the maximum probability that the states reach the set B , starting at s_0 of the MDP $\mathcal{M}(\bar{\pi}_{s, a^{(d)}, a^{(ma)}}^{(d)})$. With these definitions and notation, we can now introduce the optimization problems that the actuator meta-attacker solves.

DEFINITION 1 (ACTUATOR META-ATTACKER). *An actuator meta-attacker in the MDP \mathcal{M} wants to find the action $d^* \in \mathcal{A}^{(d)}$ they should compromise and change it to the action $a^{(ma)}$ in a given state $s^* \in \mathcal{S}$ to maximize the probability of steering the MDP to the target set B in no more than $N \in \mathbb{N}$ time instants, given that the MDP begins at a state s_0 . Thus, the meta-attacker wants to find a solution to the*

Algorithm 1 Optimal actuator meta-attack

Input: Target set B , initial state s_0 , the time constraint N , and the action that the meta-attacker applies $a^{(ma)}$.

Output: The best action d^* a meta-attacker should compromise at a state s^* to steer the MDP to the target set B after N time steps with the greatest probability.

```

1:  $d^* \leftarrow \text{None}, s^* \leftarrow \text{None}, P^* \leftarrow 0$ 
2: for each state  $s \in \mathcal{S}$  do
3:   for each  $a^{(d)} \in \mathcal{A}^{(d)}$  where  $\pi^{(d)}(a^{(d)}|s) > 0$  do
4:      $\bar{\pi}_{s,a^{(d)},a^{(ma)}}^{(d)} \leftarrow$  The modified policy (Eq. (5)).
5:      $\mathcal{M}(\bar{\pi}_{s,a^{(d)},a^{(ma)}}^{(d)}) \leftarrow$  Model of the MDP with defender's
       policy  $\bar{\pi}_{s,a^{(d)},a^{(ma)}}^{(d)}$ .
6:      $(x_{s'}^{(N)})_{s' \in \mathcal{S}} \leftarrow Pr_{\mathcal{M}(\bar{\pi}_{s,a^{(d)},a^{(ma)}}^{(d)})}^{max}(s' \models F^{\leq N} B)$ ,
7:     if  $x_{s_0}^{(N)} \geq P^*$  then
8:        $d^* \leftarrow a^{(d)}, s^* \leftarrow s, P^* \leftarrow x_{s_0}^{(N)}$ 
9: return  $d^*, s^*$ .
```

following optimization problem:

$$\begin{aligned} \max_{(s,a^{(d)}),n} \quad & Pr_{\mathcal{M}(\bar{\pi}_{s,a^{(d)},a^{(ma)}}^{(d)})}^{max}(s_0 \models F^{\leq n} B) \\ \text{s.t.} \quad & n \in \mathbb{N}, n \leq N, s_0 \in \mathcal{S} \\ & (s, a^{(d)}) \in \mathcal{S} \times \mathcal{A}^{(d)}, a^{(ma)} \in \mathcal{A}^{(d)} \end{aligned} \quad (7)$$

Sensor meta-attacker: We present the optimization problem the sensor meta-attacker wants to solve. This meta-attacker introduces a new set of states $\mathcal{S}^{(ma)}$ to the MDP graph. The sensor meta-attacker creates a new augmented MDP $\bar{\mathcal{M}}$, with graph $\bar{\mathcal{S}} = \mathcal{S}^{(ma)} \cup \mathcal{S}$ and $\bar{\mathcal{E}} \subseteq \bar{\mathcal{S}} \times \bar{\mathcal{S}}$. Let us define the set $\mathcal{E}^{(ma)} \subseteq \bar{\mathcal{E}} \times \bar{\mathcal{E}}$ that contains all the pairs of edges the meta-attacker can introduce. Let us denote the MDP

$$\bar{\mathcal{M}}(e_1, e_2)$$

to the augmented MDP where the sensor meta-attacker introduces to the MDP \mathcal{M} , the states $\mathcal{S}^{(ma)}$, the edges $(e_1, e_2) \in \mathcal{E}^{(ma)}$, with probability $p_1, p_2 \in [0, 1]$ modeling the capacity the sensor meta-attacker of creating the attack.

DEFINITION 2 (SENSOR META-ATTACKER). A sensor meta-attacker wants to introduce a state $s^* \in \mathcal{S}^{(ma)}$ and two edges $(e_1^*, e_2^*) \in \mathcal{E}^{(ma)}$ to augment the MDP \mathcal{M} to maximize the probability of steering the MDP state to the target set B from the initial state $s_0 \in \mathcal{S}$ after no more than $N \in \mathbb{N}$ time steps. Then, the meta-attacker solves the optimization problem:

$$\begin{aligned} \max_{(e_1, e_2), n} \quad & Pr_{\bar{\mathcal{M}}(e_1, e_2)}^{max}(s_0 \models F^{\leq n} B) \\ \text{s.t.} \quad & n \in \mathbb{N}, n \leq N, (e_1, e_2) \in \mathcal{E}^{(ma)}, s_0 \in \mathcal{S} \end{aligned} \quad (8)$$

Solution: We present a solution to the problem in Eq. (7) in Algorithm 1. We iterate through each possible state $s \in \mathcal{S}$ and each possible action $a^{(d)} \in \mathcal{A}^{(d)}$ where the probability the defender takes such an action in state s is not zero, i.e., $\pi^{(d)}(a^{(d)}|s) > 0$ (lines 2, 3). We then generate a model of the MDP $\mathcal{M}(\bar{\pi}_{s,a^{(d)},a^{(ma)}}^{(d)})$, where

Algorithm 2 Optimal sensor meta-attack

Input: Target set B , initial state s_0 , and time constraint N . The set of edge pairs that the sensor meta-attacker can create $\mathcal{E}^{(ma)}$ and the corresponding probability p_1, p_2 .

Output: The best edge e_1^*, e_2^* a sensor meta-attacker can introduce into the MDP to steer it to the target set B after N time steps with the greatest probability.

```

1:  $e_1^* \leftarrow \text{None}, e_2^* \leftarrow \text{None}, P^* \leftarrow 0$ .
2: for each state pair of edges in  $(e_1, e_2) \in \bar{\mathcal{E}}$  do
3:    $\bar{\mathcal{M}}(e_1, e_2) \leftarrow$  Model of the MDP with a new action that induces the transition  $e_1$  with probability  $p_1$  and another transition  $e_2$  with probability  $p_2$ .
4:    $(x_{s'}^{(N)})_{s' \in \mathcal{S}} \leftarrow Pr_{\bar{\mathcal{M}}(e_1, e_2)}^{max}(s' \models F^{\leq N} B)$ 
5:   if  $x_{s_0}^{(N)} \geq P^*$  then
6:      $e_1^* \leftarrow e_1, e_2^* \leftarrow e_2, P^* \leftarrow x_{s_0}^{(N)}$ 
7: return  $e_1^*, e_2^*$ .
```

we change the action $a^{(d)}$ to another action $a^{(ma)} \in \mathcal{A}^{(d)}$, at state $s \in \mathcal{S}$. We use the value iteration algorithm to find the probability the attacker steers the MDP to a state in the target set after N steps, beginning from s_0 , and pick the pair (s^*, d^*) that maximizes the probability.

Algorithm 2 follows a similar idea of Algorithm 1 to solve the problem in Eq. (8). We iterate over all possible edges $(e_1, e_2) \in \mathcal{E}^{(ma)}$ the meta-attacker can add. We then generate a model that includes the new states $\mathcal{S}^{(ma)}$ and edges $(e_1, e_2) \in \mathcal{E}^{(ma)}$ with probabilities p_1, p_2 into the MDP \mathcal{M} . We then use the value iteration algorithm to determine the probability the attacker steers the MDP to a state in B after N steps from state s_0 . We then get the edges $(e_1^*, e_2^*) \in \mathcal{E}^{(ma)}$ that maximize the probability the MDP is in a state $s \in B$ at step N .

Algorithm 1 provides the optimal state-action pair the meta-attacker should compromise to maximize the probability of getting control of a host in the network after a constrained amount of time. Algorithm 2 provides the optimal edges the meta-attacker should compromise to maximize the probability the attacker gets control of a target host after a constrained time. Thus, we have algorithms to solve Problem 2.

6 Experimental Setup

In this section, we present the network we use to analyze the steerability of ACDs with meta-attackers. We first present the study case and then model it using the MDP framework.

6.1 Scenario description

Several ACD studies test their agents in Cyborg[1], which is a network simulator suitable for training RL agents. Our use case is inspired by CAGE 2 [10], a popular scenario for training ACD agents with Cyborg.

For simplicity, we consider a network with three hosts and two subnetworks presented in Fig. 3. The attacker spawns in subnetwork 0 and wants to compromise host 1. The attacker needs to get



Figure 3: Study case. The attacker spawns at subnetwork 0.

root capabilities on host 0 before compromising host 1. Similarly to CAGE 2, we consider the following actions.

Vanilla Attacker actions: The attacker has the next actions:

- scan: scan subnetwork to obtain the hosts' IP addresses.
- port swipe: the attacker performs a swipe to identify the vulnerable ports of a host.
- compromise: the attacker attempts to compromise the host to get user capabilities.
- escalate: the attacker attempts to get root capabilities after obtaining user capabilities.
- sleep: the defender does nothing in one step.

Defender actions: The defender has three actions per host:

- restart: the defender restarts a host to eliminate the attacker with user capabilities.
- restore: the defender restores a host to a trusted image to eliminate an attacker with user or root capabilities.
- sleep: the defender does nothing in one step.

Meta-Attacker actions: We augment CAGE 2 with meta-attacker actions. The meta-attacker has to select only one of the following options (otherwise, the meta-attacker would be too powerful to fully disable the ACD):

- Prevent the execution of a restart
- Prevent the execution of a restore.
- Send a false alert to the ACD asking for a restart.¹

6.2 Model of the Scenario

We now present the MDP model of the previous network. Fig. 4 presents the underlying graph of the MDP. Next, we explain each element of the MDP in Eq. (1) to model the network and the actions of the defender, attacker, and meta-attacker.

Atomic propositions: Each host has four atomic propositions:

- p_0 : the attacker knows the IP of host 0.
- p_1 : the attacker knows the vulnerable port of host 0.
- p_2 : the attacker has user capabilities of host 0.
- p_3 : the attacker has root capabilities of host 0.

Similarly, we define propositions p_4, p_5, p_6 and p_7 for host 1.

States and labeling function: We model the vanilla scenario with 13 states (without meta-attacker). For each state, the labeling function L determines the information and level of compromise the attacker has on the network. The attacker begins at state s_0 , where the attacker has no information or compromise on the hosts, i.e., $L(s_0) = \{\}$. The states from s_1 to s_4 model the different levels of knowledge and access about host 0 by the attacker. The states s_5 to s_8 model the compromise of host 1 after the attacker moves laterally from host 1. Thus, the labeling function is $L(s_i) = \{p_0, \dots, p_{i-1}\}$, $i \in \{1, 2, \dots, 8\}$. So, for example, $L(s_2) = \{p_0, p_1\}$, which means the attacker knows the IP and the vulnerable ports of host 0.

¹This case models an attacker with user privileges that requires a restart to obtain root privileges but cannot force a restart by itself like in a DLL hijack.

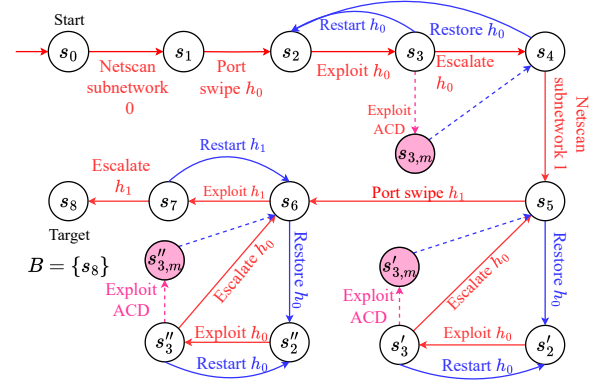


Figure 4: MDP for the two-hosts scenario in Fig. 3. Blue and red edges represent the transitions the defender and attacker induce due to their actions. Magenta circles and dotted lines model the possibility the sensor meta-attack compromises ACD. For simplicity, we omit the atomic propositions and the self-loops, modeling that the attacker's action may not succeed.

Additionally, we introduce four states: s'_2, s'_3 and s''_2, s''_3 . These states are similar to s_2 and s_3 but necessary to model the defensive actions from the ACD agent. For example, if the MDP is in the state s_5 , the attacker has root capabilities on host 0 and the IP of host 1. However, the defender may remove the attacker from host 0 by restoring the host. Then, the MDP jumps to s'_2 to model that the attacker has no control over host 0 but has information about host 1. Therefore, s_2 is not the same as s'_2 ; in this state, we have $L(s'_2) = \{p_0, p_1, p_4\} \neq L(s_2)$. We use s'_3, s''_2, s''_3 to model similar scenarios.

Vanilla defender and attacker actions: The vanilla defender action set has 5 actions, and the vanilla attacker has 9 actions. Fig. 4 shows the transitions that each agent action induces in the network. We label each edge with the action that induces the transition from one state to another. The red arrows are the attacker's possible actions, while the blue arrows correspond to the defender's actions. We omit the self-loops arrows that represent the probability the actions may not succeed.

Sensor meta-attacker actions: The sensor meta-attacker sends a false alert to the ACD asking for a restart. In the graph representation of the MDP, this can be modeled as an extra state and transition that then helps the defender move to a state with root privileges.

In our example, we introduce the states $s_{3,m}, s'_{3,m}$ and $s''_{3,m}$, which represent that the meta-attacker compromises the ACD. Then, the ACD uses the restart action to give the attacker root capabilities over host 0. These states satisfy that $L(s_{3,m}) = L(s_3)$, $L(s'_{3,m}) = L(s'_3)$, $L(s''_{3,m}) = L(s''_3)$.

Actuator meta-attacker actions: The actuator meta-attacker blocks the execution of the defender actions. We model this by eliminating a blue edge in the MDP graph representation.

Transition probability: As we mention in Section 3, this function defines the probability the MDP (or network) reaches a final state

from an initial state, given the actions. We present the transition probabilities in Appendix A.

Attacker objective: The attacker wants to compromise host 1 with root capabilities, meaning that the target set is $B = \{s_8\}$.

7 Numerical Results

In this section, we present the numerical results of applying our modeling and reachability algorithms to the network we introduced in Section 6.

The actuator meta-attacker must decide to deny the execution of an action at a state. For instance, the attacker can compromise block the restore action for host 1. This would impede the attacker from getting eliminated from host 1 once the attacker manages to compromise it. Similarly, the sensor meta-attacker must decide when to deploy the attack:

- (1) when the vanilla attacker does not have information on host 1 (state $s_{3,m}$),
- (2) when the vanilla attacker has the IP of host 1 but lost control of host 0 (state $s'_{3,m}$),
- (3) when the attacker has the IP and vulnerable port of host 1 but lost control of host 0 (state $s''_{3,m}$).

Following our discussion in Section 1, we formulate and answer three research questions.

- **RQ1:** *What is the impact of meta-attackers helping vanilla attackers?*

With this question, we want to understand if the meta-attacker can help to compromise the target host faster.

The meta-attacker speeds up the compromising process: Fig. 5 shows the effect of the meta-attacker when they use each possible strategy. When the meta-attacker compromises certain actions or sensors of the ACD, the probability the attacker compromises the target host approaches one faster. This means that the attacker can get control of a host faster if a meta-attacker is in the network.

For instance, consider the actuator meta-attacker in Fig. 5b. The meta-attacker increases the probability that the attacker compromises the target host when the meta-attacker compromises the Restore h_0 action when the attacker is trying to compromise host 1. This means that the attacker can control a host faster when the meta-attacker compromises the ACD.

The meta-attacker changes the strategy with the time constraint: Both meta-attackers change their strategy depending on the time constraint. For instance, the actuator meta-attacker prefers to compromise the restart h_1 action when the time constraint is less than 650 (see Fig. 5b). In the short term, the meta-attacker prefers to compromise the action with the highest probability of eliminating the adversary (see Appendix A). The meta-attacker takes this high risk: if they manage to compromise host 1 with user capabilities, then getting root capabilities is easier due to the meta-attacker. However, in the long term, the actuator meta-attacker prefers to compromise the restore h_0 action when the vanilla attacker is trying to compromise host 1 (state s_6). In this way, the vanilla attacker can try to compromise host 1 several times without losing the foothold of host 0 (thanks to the meta-attacker's help in blocking the restore action).

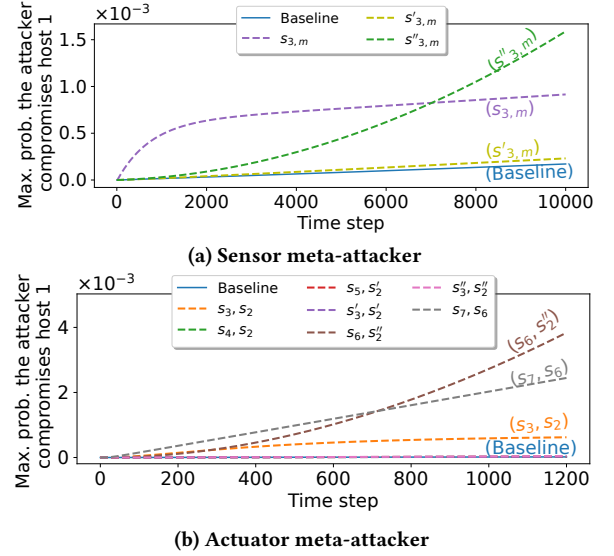


Figure 5: Probability the attacker compromises the target host when there is a meta-attacker.

Answer RQ1: The meta-attacker can help the attacker compromise a target host in the network, but the optimal meta-attack will depend on the deadline (time horizon) the attacker has.

- **RQ2:** *Can the meta-attacker follow a naive strategy to maximize the probability of compromising the target host, or do they need to use a more intelligent strategy?*

In the previous question, we wanted to understand if the meta-attacker can help the attacker compromise the target host. However, some strategies may help to compromise the target host faster than others. Thus, we want to study if the attacker needs to be strategic or if they can follow a naive strategy.

Actuator meta-attacker: A naive actuator meta-attacker might prioritize blocking the restart host 1 action, as blocking it gives the attacker the highest probability of remaining in host 1 (see Appendix A). While this strategy may be useful in the short term, Fig. 5b shows that this strategy does not lead to the maximum probability that the attacker compromises the target host in the long term. Instead, as we explained before, the optimal meta-attacker prefers to prioritize blocking the restore h_0 action when the vanilla attacker is trying to compromise host 1 (state s_6).

To further understand if the meta-attacker can follow a naive strategy, we simulate a slightly different actuator meta-attacker. This new meta-attacker can deny the execution of only one action whenever the ACD uses it. The meta-attacker now needs to decide to compromise an action among Restore h_0 , Restart h_0 , and Restart h_1 . Fig. 6a shows the optimal strategy this meta-attacker should compromise when the time constraint is $N = 2000$. For these simulations, we change the probability that the defensive actions are effective at each state. For instance, if the restart h_1 action has probability 1, the defender always eliminates an attacker with user capabilities from host 1.

A naive meta-attacker could select the action with the highest probability of eliminating the attacker from a host. Fig. 6b presents

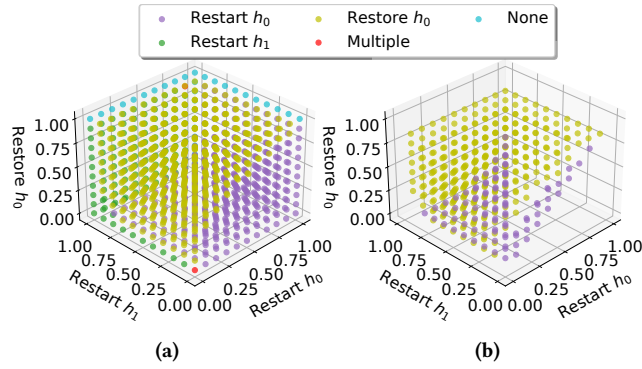


Figure 6: a) Action the optimal actuator meta-attacker target to maximize the probability of compromising host 1. b) The optimal attack that does not follow the intuition of targeting the action with the highest probability of eliminating the attacker.

cases where an optimal meta-attacker should not follow the naive strategy. In particular, when the attacker follows the non-naive strategy, the optimal meta-attacker prioritizes two scenarios: compromise Restart h_0 or Restore h_0 even if these actions have smaller success probabilities than other actions. Despite their smaller probabilities, these actions become important as the defender may use them in several stages of the game.

Sensor meta-attacker: A naive meta-attacker might choose one strategy randomly, assuming all actions have equal impact. However, Fig. 5a shows that the sensor meta-attacker should send false alerts to the ACD only when the attacker is already in host 0 and is trying to take control of host 1 (state $s''_{3,m}$) but not in any other state (e.g., at the beginning of the game).

Answer RQ2: Both meta-attackers need to be strategic to maximize the probability of compromising the target host. In several scenarios, a naive strategy does not lead to the highest probability of compromising the target host. In fact, certain strategies have almost no effect on the network.

- **RQ3.** Are there cases where the vanilla attacker requires meta-attacks in order to succeed (reach the target set)?

In the simulations from Fig. 5, the attacker can eventually compromise the target host even without a meta-attacker. We now study scenarios where the vanilla attacker cannot compromise the target host.

To study this scenario, we consider that the Restart h_1 action can always eliminate the attacker with user capabilities from host 1. Mathematically, this means the defender always produces a transition from state s_7 to s_6 . In this scenario, the vanilla attacker would not be able to compromise the target host, as the defender would always eliminate the attacker from host 1 before they can escalate to get root capabilities. However, the actuator meta-attacker could disable the Restart h_1 action allowing the adversary to get control of the target host eventually. We then conclude that the meta-attacker can allow the attacker to compromise the target host in some scenarios. We omit this time plot for space constraints.

Answer RQ3: There exists scenarios where the meta-attacker becomes necessary for the vanilla attacker to succeed. In these scenarios, the adversary cannot compromise the target host without a meta-attacker, but they can do it when there is a meta-attacker.

8 Discussion and Conclusions

Defensive AI agents may help improve the security posture of various networks, but they will also be the target of attacks. In this paper, we formulate the steerability of these AI agents by attackers, which can partially compromise the actions or the information seen by these agents during test time.

Our formulation tries to understand the capabilities that meta-attackers can gain by compromising certain parts of the ACD infrastructure. In some cases, the attacker may want to focus on compromising one action vs. another one in order to maximize the probability of reaching its target. On the defensive side, we can use this information for resource allocation to protect the network from meta-attackers.

Additionally, we found that the meta-attackers change their strategy depending on the time constraint. This suggests that defenders protecting ACDs against meta-attackers need to change their strategy depending on the network. If the ACD defends a network of drones in a short mission, we need to defend against a meta-attacker that focuses on compromising certain drones in the short term. In contrast, the defensive strategy will be different if the ACD protects an information technology (IT) network that is online for a longer time.

References

- [1] 2021. *CybORG: A Gym for the Development of Autonomous Cyber Agents*. arXiv.
- [2] Andy Applebaum, Camron Dennler, Patrick Dwyer, Marina Moskowicz, Harold Nguyen, Nicole Nichols, Nicole Park, Paul Rachwalski, Frank Rau, Adrian Webster, et al. 2022. Bridging automated to autonomous cyber defense: Foundational analysis of tabular q-learning. In *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security*. 149–159.
- [3] Christel Baier and J P Katoen. 2008. *Principles of model checking*. MIT press.
- [4] Elizabeth Bates, Vasilios Mavroudis, and Chris Hicks. 2023. Reward Shaping for Happier Autonomous Cyber Security Agents. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*. 221–232.
- [5] Robert G Campbell, Magdalini Eirinaki, and Younghee Park. 2023. A Curriculum Framework for Autonomous Network Defense using Multi-agent Reinforcement Learning. In *2023 Silicon Valley Cybersecurity Conference (SVCC)*. IEEE, 1–8.
- [6] Edmund M Clarke, Thomas A Henzinger, Helmut Veith, Roderick Bloem, et al. 2018. *Handbook of model checking*. Vol. 10. Springer.
- [7] Myles Foley, Mia Wang, Chris Hicks, Vasilios Mavroudis, et al. 2023. Inroads into autonomous network defence using explained reinforcement learning. *arXiv preprint arXiv:2306.09318* (2023).
- [8] Kim Hammar and Rolf Stadler. 2022. Intrusion prevention through optimal stopping. *IEEE Transactions on Network and Service Management* 19, 3 (2022), 2333–2348.
- [9] Yi Han, David Hubcenko, Paul Montague, Olivier De Vel, Tamas Abraham, Benjamin IP Rubinstein, Christopher Leckie, Tansu Alpcan, and Sarah Erfani. 2020. Adversarial reinforcement learning under partial observability in autonomous computer network defence. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.
- [10] Mitchell Kiely, David Bowman, Maxwell Standen, and Christopher Moir. 2023. On Autonomous Agents in a Cyber Defence Environment. *preprint arXiv:2309.07388* (2023).
- [11] Alexander Kott, Paul Theron, Martin Drašar, Edlira Dushku, Benoît LeBlanc, Paul Losiewicz, Alessandro Guarino, Luigi Mancini, Agostino Panico, Mauno Pihelgas, et al. 2018. Autonomous intelligent cyber-defense agent (AICA) reference architecture. Release 2.0. *arXiv preprint arXiv:1803.10664* (2018).
- [12] Ahmad Ridley. 2018. Machine learning for autonomous cyber defense. *The Next Wave* 22, 1 (2018), 7–14.
- [13] Maria Rigaki, Ondřej Lukáš, Carlos A Catania, and Sebastian Garcia. 2023. Out of the cage: How stochastic parrots win in cyber security environments. *arXiv preprint arXiv:2308.12086* (2023).

- [14] Maxwell Standen, Martin Lucas, David Bowman, Toby J Richer, Junae Kim, and Damian Marriott. 2021. Cyborg: A gym for the development of autonomous cyber agents. *arXiv preprint arXiv:2108.09118* (2021).
- [15] Sanyam Vyas, John Hannay, Andrew Bolton, and Professor Pete Burnap. 2023. Automated cyber defence: A review. *arXiv preprint arXiv:2303.04926* (2023).
- [16] Melody Wolk, Andy Applebaum, Camron Dennler, Patrick Dwyer, Marina Moskowitz, Harold Nguyen, Nicole Nichols, Nicole Park, Paul Rachwalski, Frank Rau, et al. 2022. Beyond cage: Investigating generalization of learned autonomous network defense policies. *arXiv preprint arXiv:2211.15557* (2022).
- [17] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. 2021. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control* (2021), 321–384.
- [18] Zhengwei Zhu, Miaojie Chen, Chenyang Zhu, and Yanping Zhu. 2024. Effective defense strategies in network security using improved double dueling deep Q-network. *Computers & Security* 136 (2024), 103578.

A Defense and Attack transition probabilities

For each $a^{(a)} \in \mathcal{A}^{(a)}$, the defense actions probabilities are:

$$\begin{aligned}
 & \tilde{T}(s_2''|s_3'', (a^{(a)}, \text{restart } h_0)) = \tilde{T}(s_2'|s_3', (a^{(a)}, \text{restart } h_0)) \\
 & = \tilde{T}(s_2|s_3, (a^{(a)}, \text{restart } h_0)) = 0.5 \\
 & \tilde{T}(s_2|s_4, (a^{(a)}, \text{restore } h_0)) = \tilde{T}(s_2'|s_5, (a^{(a)}, \text{restore } h_0)) \\
 & = \tilde{T}(s_2''|s_6, (a^{(a)}, \text{restore } h_0)) = 0.3
 \end{aligned}$$

$$\tilde{T}(s_6|s_7, (a^{(a)}, \text{restart host 1})) = 0.9$$

Next, we present the attacker's actions transition probabilities. Let us define the scan subnetwork 0 as sn_0 and scan subnetwork 1 as sn_1 , the port scan host 0 as ps_0 , and port scan host 1 as ps_1 . If the defender sleeps, we consider the probabilities for the attacker's actions:

$$\begin{aligned}
 & \tilde{T}(s_1|s_0, (sn_0, \text{sleep})) = \tilde{T}(s_5|s_4, (sn_1, \text{sleep})) = 0.95 \\
 & \tilde{T}(s_2|s_1, (ps_0, \text{sleep})) = \tilde{T}(s_6|s_5, (ps_1, \text{sleep})) = 0.95
 \end{aligned}$$

Denote the exploit host 0 as ex_0 , exploit host 1 as ex_1 , and escalate host 1 as es_1 and escalate host 1 as es_1 . Then,

$$\begin{aligned}
 & \tilde{T}(s_3|s_2, (ex_0, \text{sleep})) = \tilde{T}(s_4|s_3, (es_1, \text{sleep})) = 0.006 \\
 & \tilde{T}(s_7|s_6, (ex_1, \text{sleep})) = \tilde{T}(s_8|s_7, (es_1, \text{sleep})) = 0.06
 \end{aligned}$$

Finally, denote the exploit ACD as e_acd , then

$$\begin{aligned}
 & \tilde{T}(s_{3,m}|s_3, (e_acd, \text{sleep})) = \tilde{T}(s'_{3,m}|s'_3, (e_acd, \text{sleep})) \\
 & = \tilde{T}(s''_{3,m}|s''_3, (e_acd, \text{sleep})) = 0.5
 \end{aligned}$$