ACTIVE CYBER DECEPTION AND ATTACKER INTENT RECOGNITION USING FACTORED INTERACTIVE POMDPS

by

ADITYA P. SHINDE

(Under the Direction of Prashant Doshi)

ABSTRACT

This work presents active cyber deception as a sequential decision-making problem in a two-agent context. We model the problem of cyber deception for multiple attacker types using factored finitely-nested interactive POMDPs (I-POMDP$_\mathcal{X}$). In contrast to previous work which focuses on confusing adversaries and delaying them, our approach aims to engage with the adversary to learn their intent. Formulating cyber deception as a sequential decision-making problem enables us to model multiple phases of cyber attacks on a single host. The I-POMDP$_\mathcal{X}$-based defender agent can manipulate the attacker's beliefs using decoys and false information and thus prolong the interaction to form increasingly accurate predictions of the attacker's intent. Explicit modeling of the adversary, allowed by the I-POMDP$_\mathcal{X}$, also enables us to study how deception affects the attacker's mental state. We further implement the I-POMDP$_\mathcal{X}$-based defender agent on a real honeypot system to create an adaptive high-interaction honeypot. Our experiments in both simulations and on a real honeypot show that the I-POMDP$_\mathcal{X}$-based agent performs significantly better at intent recognition than commonly used deception strategies on honeypots.

INDEX WORDS: AI-based cyber defense, Active cyber deception, Adaptive honeypots, Partially Observable MDPs, Interactive Partially Observable MDPs, I-POMDP$_\mathcal{X}$, Intent recognition

ACTIVE CYBER DECEPTION AND ATTACKER INTENT RECOGNITION USING FACTORED INTERACTIVE POMDPS

by

ADITYA P. SHINDE

B.E., University of Mumbai, India, 2016

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2020

ACTIVE CYBER DECEPTION AND ATTACKER INTENT RECOGNITION USING FACTORED INTERACTIVE POMDPS

by

ADITYA P. SHINDE

Major Professor: Prashant Doshi

Committee: Kyu Hyung Lee
Frederick Maier

Electronic Version Approved:

Ron Walcott
Interim Dean of the Graduate School
The University of Georgia
August 2020

Dedicated to my mother and father

Acknowledgements

I would like to thank my advisor, Dr. Prashant Doshi for his help and guidance with this work. His expertise and insight helped me deal with problems which otherwise would have proved very challenging. I would also like to thank Dr. Kyu Hyung Lee whose expertise and knowledge in cybersecurity helped me finish important parts of this work. I would like to thank Dr. Fred Maier for helping me throughout my MS program at UGA. Dr. Khaled Rasheed also deserves a word of mention. His machine learning course was one of the first courses I took at UGA and it motivated me to get involved in good projects and research. My colleagues at THINC lab- Omid Setayeshfar, Muhammed AbuOdeh, Christian Adkins, Hari Teja, Ehsan Asali, and Sourabh Arora also deserve thanks for their help and company.

My Mother and Father were constantly by my side throughout my journey. Their words of encouragement and belief in my abilities gave me the strength to push through some of the more stressful times in my research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Protecting modern enterprise networks against sophisticated cyber adversaries is proving to be increasingly difficult because of the evolving nature of advanced threats. Commonly used defense techniques involve the use of intrusion detection systems and other such rule-based systems. However, rule-based systems are difficult to adapt to the dynamic nature of cyber warfare. Also, rule-based systems suffer from a high false-positive rate. This makes cyber defense a challenging problem with defenders having to set up detection systems a priori and anticipate the attackers. Such a strategy gives the attackers an asymmetric advantage over defenders. The defenders have to anticipate the attackers and prepare against all possible weaknesses that the attackers may exploit. In contrast, the attackers have to only find a single flaw to bypass the defender's defenses. In response to this asymmetric advantage, a growing number of organizations are adopting cyber deception as an effective cyber defense strategy [26]. Deception provides a simpler low cost and low maintenance solution which can be easily used alongside existing rule-based systems. A key aspect of cyber deception is the use of decoy systems called *honeypots* [32]. Honeypots are identical to other hosts on the network and are equipped with additional monitoring capabilities. They are usually well camouflaged to make them indistinguishable from real hosts. Other systems in the network are oblivious to the existence of honeypots. Thus any attempt at interacting with a

honeypot can be easily flagged as suspicious activity leading to lower false-positive rates. This makes them extremely effective as tools for detecting attackers in a network. Besides, the presence of honeypots also increases uncertainty for the attackers who now have to carefully observe and choose their targets to avoid setting off alarms.

While honeypots do give the defender significant advantages, their use in the context of threat detection is a passive defense strategy. Detection and containment are certainly important aspects of cyber defense. However, these techniques do not yield any useful information about the threat. Any attempt to contain a threat immediately after detection only alerts the attacker to the presence of detection mechanisms. Instead, actively engaging with the attacker offers an opportunity to better understand the attacker's intent and capabilities. Active strategies entail adaptive deception which seeks to influence the attackers' beliefs and manipulates the attackers into performing desired actions [18]. Engaging the attacker through deception, while simultaneously protecting the critical assets offers more information about the attacker's intent, capabilities, and motivations. Such a strategy offers a proactive approach to cyber defense.

Our work applies sequential decision-making to the problem of active cyber deception. We show that AI-based deception strategies can be effective at engaging attackers for a longer duration. Longer engagements provide more information about the adversary that reveals their intent. To this end, we develop the I-POMDP$_{\mathcal{X}}$, a factored variant of the I-POMDP. This framework enables us to explicitly model the adversary in a two-agent interaction scenario. We formulate active cyber deception as a multi-agent sequential decision-making problem and use the developed I-POMDP$_{\mathcal{X}}$ framework to compute and evaluate optimal strategies.

1.1 Related work in AI-based cyber deception

Cyber deception has been an active area of research. Recently, several AI techniques are beginning to be explored for use in deception. An area of significant recent interest has been game-theoretic

multi-agent modeling of cyber deception. This approach contrasts with the decision-theoretic modeling adopted in our work. Below, we briefly mention some of the recent work related to AI-based cyber deception.

Schlenker et al. [28] apply Stackelberg games [30] to the problem of cyber deception. They introduce the *cyber deception game* as a framework to model interactions between attackers and defenders. Their work models the network reconnaissance phase of a cyber attack. They define the attacker and defender as two players in a non-cooperative game. The attacker performs network scans to learn more about the hosts in a network. The defender controls the responses to these scans. The defender responds to an attacker's scans using a mix of true and false information to deny the attacker any useful information. This significantly increases the uncertainty for the attacker. An attacker equipped with a particular exploit can no longer use network scans to locate a vulnerable host with certainty. Another similar approach by Durkota et al. [11] allocates honeypots in a network using a Stackelberg game. Given a fixed number of honeypots and real hosts, they compute the optimal randomized deployment of honeypots that maximizes the probability of an attacker interacting with a honeypot instead of a real host. Carroll et al. [5] model deception as a dynamic non-cooperative game with incomplete information. In their work, the defender chooses to camouflage a normal host as a honeypot and vice versa. During the attack, the attacker has to determine whether or not to attack a system based on its observed characteristics which can be manipulated by the defender. Jajodia et al. [17] develop a probabilistic logic to model deception during network scanning. Using this logic, they compute optimal responses that can be provided to an attacker during the network reconnaissance phase of an attack. They apply several constraints on the possible responses to ensure that these responses cause minimum damage to the defender while being consistent with the attacker's expectations. Ferguson-Walter et al. [13] model possible differences between the attacker's and defender's perceptions toward the interaction by modeling cyber deception as a hypergame [21]. Hypergames model different views of the game being played from the perspective of the players.

While the application of game theory to cyber deception provides an objective view of the interaction between attackers and defenders, most of the work in this area aims to explicitly deceive the attackers or to lead them to honeypots. In contrast, we study the interaction from the subjective perspective of a defender. The defender aims to learn the attacker's intent, and for this purpose, attempts to prolong the interaction with the attacker. We show that deception emerges naturally as an optimal strategy when the defender aims to keep the attacker engaged.

1.2 Contributions

We briefly mentioned some related work in the previous section. While those efforts focus on delaying or confusing the attackers using deception, we aim to actively engage with the attackers to learn more about their intent. To model this multi-agent interaction, we extend I-POMDPs to their factored variant, the I-POMDP$_\mathcal{X}$. The key contributions of our work are as follows:

1. We represent cyber deception on a single host as a decision-making problem between a defender and an attacker. We consult cybersecurity experts to estimate the dynamics of the environment, and to assign realistic actions to both agents.

2. We model cyber attacks in multiple phases analogous to the Lockheed Martin Cyber Kill Chain® [16] and the MITRE ATT&CK framework [33]. Also, we model multiple types of attackers with realistic objectives to simulate threats with unknown intentions.

3. We introduce a factored variant of the well-known interactive partially observable Markov decision process [14], labeled as I-POMDP$_\mathcal{X}$. I-POMDP$_\mathcal{X}$ exploits the factored structure of the problem, representing the dynamics and observation function using algebraic decision diagrams [2]. This makes the I-POMDP$_\mathcal{X}$ tractable for the cyber deception domain.

4. The I-POMDP$_\mathcal{X}$ enables explicit modeling of the opponent throughout the interaction. This gives us insight into how deception affects the attacker's subjective view of the system.

5. We test the proposed I-POMDP$_\mathcal{X}$ agent in simulations and on a testbed. We compare the I-POMDP$_\mathcal{X}$ agent against other commonly used deception strategies. Our results show that the I-POMDP$_\mathcal{X}$ based agent performs significantly better at intent recognition than other baseline strategies.

6. Lastly, we implement the attacker and the defender on a real system to show that our deception techniques and strategies can be realistically implemented on honeypots.

Through our work, we hope to motivate a more proactive approach towards cyber defense in contrast to the reactive strategies currently being used.

1.3 Structure of this work

In this section, we introduced cyber deception and briefly reviewed some work on AI-based deception. We also mentioned how our work differs from previous work and some of our key contributions. Below we briefly mention the contents of the subsequent chapters.

Chapter 2 provides some background on sequential decision-making frameworks. Our work formulates cyber deception as a sequential decision-making problem. Hence this background lays the groundwork for the next chapters in which we introduce the I-POMDP$_\mathcal{X}$. We review factored POMDPs and their solution techniques. We then discuss finitely-nested I-POMDPs which extend POMDPs to multi-agent settings.

Chapter 3 introduces the I-POMDP$_\mathcal{X}$ framework. We apply the techniques reviewed in Chapter 2 to extend finitely nested I-POMDPs to their factored variant. We show how I-POMDP$_\mathcal{X}$ factors can be represented using ADDs analogous to factored POMDPs. We also define the cyber deception domain using the I-POMDP$_\mathcal{X}$ and define the dynamics for attacker and defender agents.

In *Chapter 4* we define the different attacker types which we use to evaluate the defender. We discuss their optimal policies and their objectives.

Chapter 5 shows the results of our experiments. We evaluate the I-POMDP$_{\mathcal{X}}$-based defender in simulations and on a real testbed. This chapter shows the duration of the interactions and the performance of intent recognition in both cases.

Chapter 6 concludes the thesis. We briefly highlight our contributions and also mention some limitations of our approach. We also briefly discuss possible areas of improvement.

Chapter 2

Background

We model the interaction between the cyber attacker and defender as an interaction between two self-interested agents. The attacker's problem is modeled as a POMDP. The defender who also models the attacker explicitly is modeled as a finitely nested I-POMDP. In this chapter, we provide some background on factored POMDPs and finitely nested I-POMDPs. We show how compact representations and approximate solution techniques are used to solve factored POMDPs efficiently. In later chapters, we will use the same principles to achieve computational advantages in solving I-POMDPs.

2.1 Factored POMDPs

Partially Observable MDPs (POMDPs) are a generalization of MDPs. POMDPs model sequential decision-making problems for partially observable environments [20]. In practice, POMDPs often exhibit structure in their transition and observation functions. States and observations in such cases influence, and are in turn influenced by a select few states. Hence the POMDP can be compactly represented as a *dynamic Bayesian network* (DBN) [6]. Accordingly, the transition and observation functions take the form of conditional probability distributions over sets of variables in the

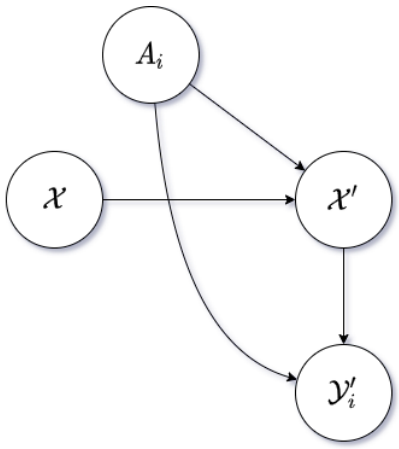

Figure 2.1: A factored POMDP as a two-slice DBN. A_i represents agent i's action. $\mathcal{X}$ and $\mathcal{X}'$ are the sets of pre-action and post-action state variables. $\mathcal{Y}'_i$ is the set of observation variables.

DBN. POMDPs which exhibit such structure are called factored POMDPs [15, 19, 27]. DBNs explicitly represent *conditional independence* [24] between variables. This is an important property that enables compressed representations of POMDP dynamics using conditional probability distributions. Figure 2.1 shows the DBN representation of factored POMDP dynamics. Below, we provide a brief background on factored POMDP representations and solution techniques and how they relate to their unfactored counterparts.

2.1.1 Definition

Similar to POMDPs, a factored POMDP can be defined as the tuple,

$$\text{POMDP}_i = \langle \mathcal{X}, A_i, T_i, \mathcal{Y}_i, O_i, \mathcal{R}_i \rangle$$

where,

- $\mathcal{X} = \{X_1, X_2, X_3, ..., X_n\}$ is the state space consisting of n state variables. Every X_i is a single state variable. A state is represented by a joint instantiation of all state variables.

- A_i is the set of actions available to agent i.
- T_i represents the transition function, T_i: $S \times A \times S \rightarrow [0, 1]$. The transition function is represented in the DBN as the conditional probability distribution $P(\mathcal{X}'|\mathcal{X}, A_i)$
- $\mathcal{Y}_i = \{Y_{i_1}, Y_{i_2}, Y_{i_3}, ..., Y_{i_n}\}$ is the observation space. Similar to the state variables, every Y_{i_i} is an observation variable.
- O_i is the observation function, O_i: $S \times A \times \Omega \rightarrow [0, 1]$. The conditional probability distribution $P(\mathcal{Y}'_i|\mathcal{X}', A_i)$ denotes the observation function in a factored POMDP
- $\mathcal{R}_i$ defines the reward function for agent i, $\mathcal{R}_i$: $\mathcal{X} \times A_i \rightarrow \mathbb{R}$.

2.1.2 Factor representation using ADDs

In section 2.1, we mentioned how conditional independence can be leveraged to compress the transition and observation functions. Additionally, the dynamics of a factored POMDP can be represented more compactly by exploiting *context-specific independence* [3]. Context-specific independence is the property that some random variables are independent of each other for specific values of those variables. Boutilier and Poole have previously used decision trees (DTs) to leverage context-specific independence to represent factors [4]. Similarly, Hoey et al., and Feng and Hansen use algebraic decision diagrams (ADDs) to represent MDP and POMDP factors respectively [19, 12]. ADDs are extensions of binary decision diagrams (BDDs) which are commonly used for boolean function verification [2]. In probabilistic planning, ADDs are used to represent probability functions. A significant advantage that ADDs offer over DTs are that they are canonical– every probability function has a unique ADD representation that cannot be compressed further. ADDs are essentially DTs with merged isomorphic sub-trees. Formally, an ADD representing a factor F consisting of discrete random variables $X_1, X_2, ..., X_n$ can be denoted as,

$$F = C, \quad \text{if } F \text{ is a terminal node}$$

$$F(X_1, X_2, ..., X_n) = F_{x_1}(X_2, ..., X_n), \quad \text{where } X_1 = x_1$$

Here, $C \in \mathbb{R}$ is the value of the leaf node which represents the probability value for a given joint instantiation of the random variables. The term x_1 denotes a given value of the random variable X_1, and F_{x_1} is the subgraph rooted at the node at the x_1 edge of the subgraph rooted at X_1.

Feng and Hansen have shown how ADDs can be used to represent the dynamics of a factored POMDP [12]. For a given action $A_i = a_i$, they define the transition function for all post-action variables as the ADD $P^{a_i}(\mathcal{X}'|\mathcal{X})$. The terms $\mathcal{X}'$ and $\mathcal{X}$ are as defined in figure 2.1 and section 2.1.1. This factor represents the *complete action diagram* for factored POMDPs analogous to the one defined for factored MDPs by Hoey et al. [19]. Similarly, the observation function is denoted using the ADD $P^{a_i}(\mathcal{Y}'_i|\mathcal{X}')$. This is defined as a *complete observation diagram*. Using the complete action diagram and the complete observation diagram, we can define an ADD containing all transition probabilities for action $A_i = a_i$ and observation $\mathcal{Y}_i = o_i$ as,

$$P^{a_i,o_i}(\mathcal{X}'|\mathcal{X}) = P^{a_i}(\mathcal{X}'|\mathcal{X})P^{a_i}(\mathcal{Y}'|\mathcal{X}') \tag{2.1}$$

The factor $P^{a_i,o_i}(\mathcal{X}'|\mathcal{X})$ is analogous to $P(o_i, s'|s, a_i)$, $P^{a_i}(\mathcal{X}'|\mathcal{X})$ to $P(s'|s, a_i)$, and $P^{a_i}(\mathcal{Y}'|\mathcal{X}')$ to $P(o_i|s', a_i)$ in unfactored POMDPs.

2.1.3 Belief Update

Since the agent is unable to observe the state perfectly, it maintains a probability distribution over all possible states. This distribution is called a *belief* and it summarizes the agent's entire ac-

tion/observation history [1]. The belief update for a POMDP can be written as,

$$\begin{aligned} b_i^t(s^t) &= P(s^t|o_i^{t-1}, a_i^{t-1}, b_i^{t-1}) \\ &= \beta O_i(s^t, a_i^{t-1}, o_i^t) \sum_{s^{t-1} \in S} T_i(s^t, a_i^{t-1}, s^{t-1}) b_i^{t-1}(s^{t-1}) \end{aligned} \tag{2.2}$$

where β is the normalization factor, b_i^t is the belief of agent i at interaction time step t, and O_i and T_i are the defined in section 2.1.1.

For a factored POMDP, a belief is represented as the joint distribution of all state variables. The belief update can be written in terms of factors as,

$$b_i^{a_i,o_i}(\mathcal{X}') = \beta \sum_{\mathcal{X}} b(\mathcal{X}) P^{a_i,o_i}(\mathcal{X}'|\mathcal{X}) \tag{2.3}$$

where $b(\mathcal{X})$ is the prior belief, $P^{a_i,o_i}(\mathcal{X}'|\mathcal{X})$ is the factor derived in equation 2.1 from the complete action diagram and the complete observation diagram.

2.1.4 Solutions

An optimal POMDP policy maps beliefs of the agent to optimal actions that should be performed in a particular belief state to get the maximum expected reward: $\pi^* : \mathcal{B}_i \rightarrow \Delta(A_i)$. Here, $\mathcal{B}_i$ is the set of all possible belief points for agent i and $\Delta(A_i)$ is the distribution over optimal actions for a particular belief point $b \in \mathcal{B}_i$. Below, we provide some context on approximate POMDP solutions and how ADDs are for solving factored POMDPs.

The optimal policy is obtained from the value function which maps belief states to real numbers representing the preferences of the agent: $V : \mathcal{B}_i \rightarrow \mathbb{R}$. The value function for POMDPs is defined as,

$$V(b) = \max_{a_i \in A_i} \left\{ \rho(b, a_i) + \gamma \sum_{o_i \in O_i} P(o_i|b, a_i) V_{t-1}(\tau(b, o_i, a_i)) \right\} \tag{2.4}$$

where τ represents the belief update mentioned in the equation 2.2,

$\rho(b, a_i) = \sum_{s \in S} R(s, a_i) b(s)$, and $P(o_i|b, a_i) = \sum_{s,s' \in S} b(s) P(s'|s, a_i) P(o_i|s', a_i)$

Equation 2.4 forms the basis for *value iteration* techniques for exact solutions to POMDPs [20, 31, 23]. The optimal value function at horizon t can thus be computed from the previous value function at $t-1$ by repeatedly performing the dynamic programming backup operation in equation 2.4. The backup operation maintains the piecewise linearity of the value function [31]. Hence, the value function can be represented using a set of vectors as, $V = \{v_0, v_1, ..., v_n\}$ where each vector v_i has the same dimensions as the state space. These are called α vectors. The value function can now be defined in the form of α vectors as,

$$V_t(b) = \max_{\alpha \in \Gamma_t} b \cdot \alpha$$

where Γ_t is the set of all α vectors at time step t.

Exact solutions to POMDPs can often prove to be computationally intractable. POMDPs suffer from the *curse of dimensionality* arising from the dimensionality of the state space, and also the *curse of history*, which depends on the planning horizon of the problem [20]. As a faster alternative to exact solutions, approximate solution techniques such as *point-based value iteration* (PBVI) are commonly used to solve larger POMDPs. Pineau et. al. propose an anytime PBVI algorithm that provides fast approximate solutions to larger POMDPs [25]. They use a fixed set of belief points and update the value function at those points during backup operations. In addition, they modify the backup operation to maintain a single α vector for each belief point. In this way, the size of the value function is bounded by the size of the set of belief points. Using the modified backup operation, they define the steps for generating the set of α vectors Γ_t from the previous α vectors

Γ_{t-1} as,

$$\begin{aligned}
\Gamma_t^{a,*} &\leftarrow \alpha^{a,*}(s) = R(s,a) \\
\Gamma_t^{a,o_i} &\leftarrow \alpha_i^{a,o_i}(s) = \gamma \sum_{s' \in S} T(s,a,s') O(s',a,o_i) \alpha_i(s'), \quad \forall \alpha_i \in \Gamma_{t-1} \\
\Gamma_t^a &\leftarrow \alpha_b^a = \Gamma_t^{a,*} + \sum_{o_i \in \Omega_i} \arg\max_{\alpha \in \Gamma_t^{a,o_i}} \left(\sum_{s \in S} \alpha(s) b(s) \right), \quad \forall b \in B \\
\alpha_b &= \arg\max_{\Gamma_t^a, \forall a \in A} \left(\sum_{s \in S} \Gamma_t^a(s) b(s) \right) \\
\Gamma_t &= \bigcup_{b \in B} \alpha_b
\end{aligned} \tag{2.5}$$

For factored POMDPs, ADDs are used to represent the value function compactly as $\mathcal{V} = \{v_0(\mathcal{X}), v_1(\mathcal{X}), ...v_n(\mathcal{X})\}$ [12]. The backup operation shown in equation 2.5 can be rewritten for factored representations as follows,

$$\begin{aligned}
\Gamma_t^{a_i,*} &\leftarrow \alpha^{a_i,*}(\mathcal{X}) = \mathcal{R}^{a_i}(\mathcal{X}) \\
\Gamma_t^{a_i,o_i} &\leftarrow \alpha_i^{a_i,o_i}(\mathcal{X}) = \gamma \sum_{\mathcal{X}'} P^{a_i,o_i}(\mathcal{X}'|\mathcal{X}) \alpha_i(\mathcal{X}'), \quad \forall \alpha_i \in \Gamma_{t-1} \\
\Gamma_t^{a_i} &\leftarrow \alpha_b^{a_i} = \Gamma_t^{a_i,*} + \sum_{o_i \in \mathcal{Y}_i} \arg\max_{\alpha \in \Gamma_t^{a_i,o_i}} \left(\sum_{\mathcal{X}} \alpha(\mathcal{X}) b(\mathcal{X}) \right), \quad \forall b \in B \\
\alpha_b &= \arg\max_{\Gamma_t^{a_i}, \forall a_i \in A_i} \left(\sum_{\mathcal{X}} \Gamma_t^{a_i}(\mathcal{X}) b(\mathcal{X}) \right) \\
\Gamma_t &= \bigcup_{b \in B} \alpha_b
\end{aligned} \tag{2.6}$$

Above, the reward function for action a_i is represented as the ADD $\mathcal{R}^{a_i}(\mathcal{X})$. POMDP reward functions exhibit the property of *additive separability* [34]. Additive separability is the property that a reward function for a particular action a_i can be built from smaller reward functions over individual state variables: $\mathcal{R}^{a_i}(\mathcal{X}) = \sum_{i=0}^{n} \mathcal{R}^{a_i}(X_i)$. ADDs allow compact representations of separable reward functions.

To further improve the time complexity of POMDP solutions, Vlassis and Spaan have proposed

a technique that randomly samples belief points from the belief set and updates the value function at those points [35]. Their technique, called *Perseus*, selects successively smaller subsets of beliefs and provides faster solutions to large POMDPs. The *Symbolic Perseus* solver augments the Perseus technique with ADDs for solving factored POMDPs [27]. In practice, the symbolic Perseus solver is augmented with further enhancements. ADD computations are cached and the results are looked up to save on computations. Further, ADDs are approximated to decrease their size. Also, the solution is only computed over observations with higher probabilities. These approximations have been shown to cause a negligible decrease in solution quality.

2.2 Finitely nested Interactive POMDPs

Interactive POMDPs (I-POMDPs) generalize POMDPs to sequential decision-making in multi-agent environments [14, 8]. I-POMDPs can explicitly model other agents in a multi-agent interaction. In our work, we use a factored variant of finitely nested I-POMDPs which we call I-POMDP$_\mathcal{X}$ to model the defender. This section provides some context on finitely nested I-POMDPs that will help us define I-POMDP$_\mathcal{X}$ in a later chapter.

2.2.1 Definition

Formally, an I-POMDP for agent i in a two-agent environment is defined as,

$$\text{I-POMDP}_i = \langle IS_i, A, T_i, \Omega_i, O_i, R_i \rangle$$

where,

- IS_i denotes the interactive state space. The IS_i includes the physical state S and models of the other agent M_j. Models of the other agent can be intentional or subintentional [7]. In our work, we assume intentional models for the opponent as they model the opponent's beliefs

and capabilities as a rational agent.

- $A = A_i \times A_j$ is the set of joint actions of both agents.
- T_i represents the transition function, T_i: $S \times A \times S \rightarrow [0, 1]$. The transition function is defined only over the physical state space S. This is a consequence of the model non-manipulability assumption – an agent's actions do not directly influence the other agent's models [14, 8].
- Ω_i is the set of agent i's observations.
- O is the observation function, O_i: $S \times A \times \Omega \rightarrow [0, 1]$. Similar to T_i, the observation function is defined only over S as a consequence of the model non-observability assumption – other's model parameters may not be observed directly [14, 8].
- R_i defines the reward function for agent i, R_i: $S_i \times A \rightarrow \mathbb{R}$.

The ability to explicitly model other agents gives rise to the possibility of infinite nesting due to recursive modeling. However, for such cases, the I-POMDP cannot be computed [8]. Hence, in practice the level of nesting is fixed to level 0 and the interactive state space is built bottom up. I-POMDPs with finite levels of nesting are called finitely nested I-POMDPs. The interactive state space $IS_{i,l}$ for a finitely nested I-POMDP at strategy level l is defined bottom up as,

$$
\begin{aligned}
IS_{i,0} &= S, && \Theta_{j,0} = \{\langle b_{j,0}, \hat{\theta}_j \rangle : b_{j,0} \in \Delta(IS_{j,0})\} \\
IS_{i,1} &= S \times M_{j,0}, && \Theta_{j,1} = \{\langle b_{j,1}, \hat{\theta}_j \rangle : b_{j,1} \in \Delta(IS_{j,1})\} \\
&\vdots \\
IS_{i,l} &= S \times M_{j,l-1}, && \Theta_{j,l} = \{\langle b_{j,l}, \hat{\theta}_j \rangle : b_{j,l} \in \Delta(IS_{j,l})\}.
\end{aligned}
$$

Above, $\hat{\theta}_j$ represents agent j's frame, defined as $\hat{\theta}_j = \langle A_j, \Omega_j, T_j, O_j, R_j, OC_j \rangle$. Here, OC_j represents j's optimality criterion and the other terms are as defined previously. Θ_j is the set of agent j's intentional models, defined as $\theta_j = \langle b_j, \hat{\theta}_j \rangle$. At level $l > 0$, agent i can have infinite models of

agent j in the interactive state space. To maintain computability, the models of agent j are limited to a finite set, $\Theta_{j,l-1}$ [9]. The set of j's models is updated after every interaction to account for the belief update of agent j. $IS_{i,l}$ can be then defined as,

$$IS_{i,l} = S \times \mathsf{Reach}(\Theta_{j,l-1}, H), \quad \Theta_{j,l} = \{\langle b_{j,l}, \hat{\theta}_j \rangle : b_{j,l} \in \Delta(IS_{j,l})\}.$$

Here, $\mathsf{Reach}(\Theta_{j,l-1}, H)$ is the set of level $l-1$ models that j could have in H steps; Reach $(\Theta_{j,l-1}, 0) = \Theta_{j,l-1}$. We obtain $\mathsf{Reach}()$ by repeatedly updating j's beliefs in the models in $\Theta_{j,l-1}$.

2.2.2 Belief Update

Similar to POMDPs, an agent's beliefs in I-POMDPs are sufficient statistics [14, 10]. An I-POMDP agent maintains beliefs over the physical state space S and also the models of the other agent, M_j. The belief update for I-POMDPs recursively updates the nested beliefs at all strategy levels. It is defined as,

$$\begin{aligned} P(is_t) = \beta \sum_{is^{t-1}} P(is^{t-1}) \sum_{a_j^{t-1}} P(a_j^{t-1}|\theta_j^{t-1}) O_i(s^t, at-1, o_i^t) \\ \times T(s^{t-1}, a^{t-1}, s^t) \sum_{o_j^t} \tau_{\theta_j^t}(b_j^{t-1}, a_j^{t-1}, o_j^t, b_j^t) O_j(s^t, a^{t-1}, o_j^t) \end{aligned} \tag{2.7}$$

Above, the term $\tau_{\theta_j^t}$ represents agent j's belief update, $P(a_j^{t-1}|\theta_j^{t-1})$ is $\frac{1}{|\text{OPT}(A_j)|} \forall a_j \in \text{OPT}(A_j)$ and 0 for other actions. OPT(A_j) is the set of all optimal actions for agent j for type θ_j^{t-1}.

2.2.3 Solutions

Similar to POMDPs, I-POMDP value functions map an agent's beliefs to real numbers representing the preferences of the agent: $V : \mathcal{B} \rightarrow \mathbb{R}$. The value function for I-POMDPs is defined as,

$$
\begin{aligned}
U(\theta_j) &= \max_{a_i \in A_i} Q(\theta_i, a_i) \\
U(\theta_j) &= \max_{a_i \in A_i} \left\{ \sum_{is} ER(is, a_i) b_i(is) + \gamma \sum_{o_i \in \Omega_i} P(o_i | a_i, b_i) U(\langle SE_{\theta_i}(b_i, a_i, o_i), \hat{\theta}_i \rangle) \right\}
\end{aligned} \tag{2.8}
$$

where,

$$ER_i(is, a_i) = \sum_{a_j} R_i(is, a_i, a_j) P(a_j | m_j)$$

The set of optimal actions for agent i at infinite horizon with discounting for a particular belief is given by,

$$OPT(\theta_i) = \arg\max_{a_i \in A_i} Q(\theta_i, a_i)$$

Value iteration in I-POMDPs is based on equation 2.8. I-POMDP value functions can also be expressed in terms of alpha vectors as,

$$V_t(\langle b_{i,l}, \hat{\theta}_i \rangle) = \max_{\alpha \in \Gamma_t} b \cdot \alpha$$

where the belief b is defined over the interactive state space $IS_{i,l}$ and the α vectors is a function of $IS_{i,l}$ written as $\alpha(is_{i,l})$.

Like POMDPs, I-POMDPs also suffer from the curse of dimensionality and the curse of history. In addition, I-POMDPs suffer from the curse of history of the modeled opponent [9]. Hence, exact solutions to I-POMDPs can be computationally intractable for larger problems. Approximate solutions can be much faster and yield a lot of computational savings. Interactive point based value iteration is one such technique which is analogous to PBVI for POMDPs [9]. Just like in PBVI, I-PBVI operates on a finite set of interactive belief points. Hence, the size of the solution is bounded.

The backup operation for generating the next set of alpha vectors, Γ_t from the previous set Γ_{t-1} for I-POMDPs is,

$$\Gamma_t^{a_i,*} \leftarrow \alpha^{a_i,*}(is) = \sum_{a_j \in A_j} R_i(s, a_i, a_j) P(a_j|\theta_{j,l-1})$$

$$\Gamma_t^{a_i,o_i} \leftarrow \alpha_i^{a_i,o_i}(is) = \gamma \sum_{is'} \sum_{a_j} P(a_j|\theta_{j,l-1}) T_i(s, a_i, a_j, s') O_i(s', a_i, a_j, o_i)$$

$$\times \sum_{o_j} O_j(s', a_i, a_j, o_j) \delta_D(SE(b_{j,l-1}, a_j, o_j) - b'_{j,l-1}) \alpha_i(is'), \forall \alpha_i \in \Gamma_{t-1} \quad (2.9)$$

$$\Gamma_t^{a_i} \leftarrow \Gamma_t^{a_i,*} + \sum_{o_i \in \Omega_i} \underset{\alpha \in \Gamma_t^{a_i,o_i}}{\arg\max} \left(\sum_{is \in IS} \alpha(is) b(is) \right), \quad \forall b_{i,l} \in B_{i,l}$$

$$\mathcal{V}^t \leftarrow \underset{\alpha^t \in \bigcup_{a_i} \Gamma^{a_i}}{\arg\max} (\alpha \cdot b_{i,l}), \quad \forall b_{i,l} \in B_{i,l}$$

Here, $\mathcal{V}^t$ contains at the most $|B_{i,l}|$ α vectors [9].

2.3 Summary

POMDPs generalize MDPs to partially observable settings and are widely used in probabilistic planning. In practice, POMDP states and observation can be aggregated and the dynamics can be compactly represented using *dynamic Bayesian networks* (DBNs). The resulting variant is called a *factored POMDP*. DBNs enable compact representations of POMDP dynamics as conditional probability distributions. These representations can be further compressed using *algebraic decision diagrams* (ADDs). Depending on the level of state abstraction, factored POMDPs can provide computational improvements over POMDPs. However, due to the *curse of dimensionality* and the *curse of history*, even factored POMDPs are unable to scale to larger problems. For fast and scalable solutions to POMDPs, approximate solution techniques like *point-based value iteration* (PBVI) are used. The PBVI method performs the value function backup over individual belief points and thus limits the size of the approximated value function. The *Perseus* algorithm further

improves on PBVI by using a subset of belief points to perform the backup operation. The *symbolic Perseus* technique combines ADDs with Perseus to solve large POMDPs efficiently.

Interactive POMDPs (I-POMDPs) extend POMDPs to multi-agent settings. I-POMDPs explicitly model the opponent in an interaction. Thus, the state space of an I-POMDP contains the models of the other agent in addition to the physical states. However, possibly infinite levels of nesting and an infinite number of opponent's models can make I-POMDPs impossible to compute. Hence the level of nesting is fixed and the interactive state space is built bottom-up from level 0. Also, the opponent's models are limited to a finite set which is updated after every interaction. I-POMDPs with finite levels of nesting are called finitely nested I-POMDPs. The same problems which make exact solutions to POMDPs intractable also affect I-POMDPs. Also, due to the explicit modeling of the opponent, I-POMDPs suffer from the curse of history of the opponent. For faster solutions to I-POMDPs, approximate solution techniques are used. One such technique is *interactive point-based value iteration* (IPBVI). IPBVI is analogous to PBVI in POMDPs. Similar to PBVI, IPBVI performs the backup operation at select belief points to keep the size of the approximated value function bounded.

Chapter 3

I-POMDP$_\mathcal{X}$ for cyber deception

In this chapter, we present a factored variant of the finitely nested I-POMDP, the I-POMDP$_\mathcal{X}$. We discuss I-POMDP$_\mathcal{X}$ solution techniques and factor representations. Subsequently, we model active cyber deception as an I-POMDP$_\mathcal{X}$ and explain the domain in detail.

3.1 Finitely nested factored I-POMDPs (I-POMDP$_\mathcal{X}$)

In section 2.1 we reviewed how factored POMDPs coupled with approximate solution techniques are effective for solving large POMDPs. Now, we apply the same principles to I-POMDPs reviewed in section 2.2 to create a factored variant, the I-POMDP$_\mathcal{X}$. In addition, we restrict the nesting levels of I-POMDP$_\mathcal{X}$ and also the set of agent j's models similar to finitely nested I-POMDPs to maintain computability.

3.1.1 Definition

Formally, the I-POMDP$_\mathcal{X}$ is defined as:

$$\text{I-POMDP}_\mathcal{X} = \langle \mathcal{IS}_i, A, T_i, \mathcal{Y}_i, O_i, \mathcal{R}_i \rangle$$

- $\mathcal{IS}_i$ is the factored interactive state space. This consists of the set of physical state variables $\mathcal{X} = \{X_1, X_2, ..., X_n\}$ and the set of agent j's models $M_j = \{mj_1, mj_2, ..., mj_n\}$. In a finitely-nested I-POMDP$_\mathcal{X}$ the set M_j is bounded similarly to finitely-nested I-POMDPs.

- The joint action set A is defined exactly as in finitely nested I-POMDPs.

- T_i defines the transition function represented using ADDs as $P^{a_i,a_j}(\mathcal{X}'|\mathcal{X})$ for $a_i \in A_i$ and $a_j \in A_j$. The model non-manipulability assumption applies to I-POMDP$_\mathcal{X}$.

- $\mathcal{Y}_i = \{Y_{i_1}, Y_{i_2}, ..., Y_{i_n}\}$ is the set of observation variables which make up the observation space for agent i.

- O_i is the observation function represented as ADDs, $P^{a_i,a_j}(\mathcal{Y}_i'|\mathcal{X}')$. The model non-observability assumption also applies to I-POMDP$_\mathcal{X}$.

- $\mathcal{R}_i$ defines the reward function for agent i. The reward function is represented as the ADD, $\mathcal{R}^{a_i,a_j}(\mathcal{X})$.

We use algebraic decision diagrams (ADDs) [2] to represent the factors for agent i's transition, observation, and reward functions compactly. Figure 3.1 shows the two-slice DBN representation of the I-POMDP$_\mathcal{X}$.

3.1.2 I-POMDP$_\mathcal{X}$ dynamics using ADDs

We now show how the ADDs representing the I-POMDP$_\mathcal{X}$ dynamics are obtained from the transition and observation functions. As shown in figure 3.1, $\mathcal{X} = \{X_1, ..., X_n\}$ and $\mathcal{X}' = \{X_1', ..., X_n'\}$ are the sets of pre-action and post-action physical state variables. $\mathcal{Y}_i' = \{Y_{i_1}', ..., Y_{i_n}'\}$ and $\mathcal{Y}_j' = \{Y_{j_1}', ..., Y_{j_n}'\}$ denote the sets of observation variables for agents i and j respectively. Analogous to the *complete action diagram* for factored POMDPs reviewed in section 2.1.2 and for factored MDPs [19], the ADD $P^{a_i}(\mathcal{X}'|\mathcal{X}, A_j) = P^{a_i}(X_1'|X_2', \ldots, X_n', \mathcal{X}, A_j) \times ... \times P^{a_i}(X_n'|\mathcal{X}, A_j)$ represents the complete action diagram for action $A_i = a_i$. Similarly, the ADD $P^{a_i}(\mathcal{Y}_i'|\mathcal{X}', A_j) =$

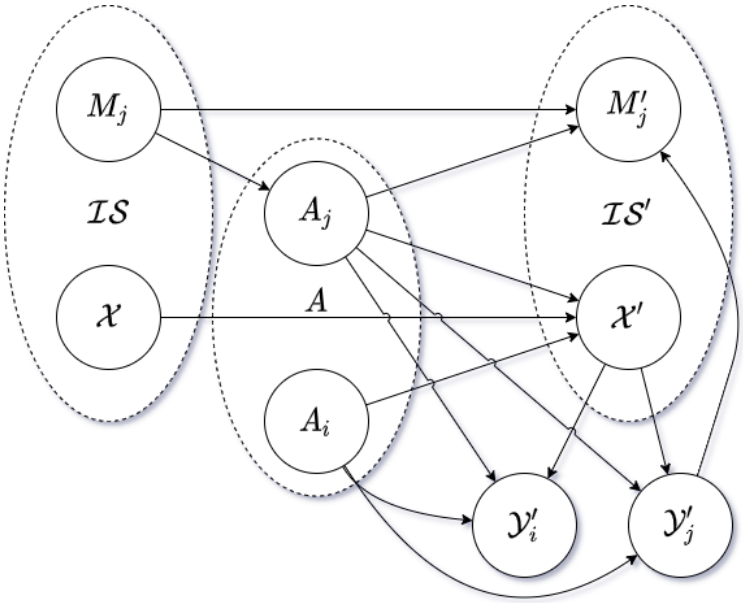

Figure 3.1: A two-slice DBN representation of I-POMDP$_\mathcal{X}$. The factored interactive state space $\mathcal{IS}$ consists of the factored physical state space $\mathcal{X}$ and the set of agent j's models, M_j. $\mathcal{Y}_i$ and $\mathcal{Y}_j$ make up agent i and agent j's observation space. Similarly, A_i and A_j denote the actions of both agents.

$P^{a_i}(Y'_{i_1}|\mathcal{X}', A_j) \times ... \times P^{a_i}(Y'_{i_n}|\mathcal{X}', A_j)$ represents the complete observation function which is analogous to the *complete observation diagram* [12]. The factored interactive state space $\mathcal{IS}$ contains models of agent j in addition to the physical states $\mathcal{X}$. In a finitely nested I-POMDP$_\mathcal{X}$, the set of models of j is limited to those contained in $\mathsf{Reach}(\Theta_{j,l-1}, H)$. This set is updated after every interaction. We define $M_j = \{m_{j_1} : \langle b_{j_1}, \hat{\theta}_{j_1}\rangle, ..., m_{j_n} : \langle b_{j_q}, \hat{\theta}_{j_r}\rangle\}$ as the set of all models in $\mathsf{Reach}(\Theta_{j,l-1}, H)$. Neither a_j nor o_j are directly accessible to agent i. Hence they are represented as conditional probability distributions using ADDs $P(A_j|M_j)$ and $P^{a_i}(\mathcal{Y}'_j|\mathcal{X}', A_j)$. The I-POMDP belief update recursively updates the beliefs for models of agent j. In I-POMDP$_\mathcal{X}$, this update is captured in the conditional distribution over M'_j as $P^{a_i}(M'_j|M_j, \mathcal{Y}'_j, A_j, \mathcal{X}') =$ $P^{a_i}(M'_j|M_j, A_j, \mathcal{Y}'_j) \times P^{a_i}(\mathcal{Y}'_j|\mathcal{X}', A_j)$. Using these factors, we can now define the joint transition

function for $\mathcal{X}'$ and M_j' given action a_i and observation o_i as a single ADD,

$$\begin{aligned} P^{a_i,o_i}(M_j', \mathcal{X}' | M_j, \mathcal{X}) &= \sum_{A_j, \mathcal{Y}_j'} P^{a_i,o_i}(\mathcal{Y}_j', M_j', \mathcal{X}', A_j | M_j, \mathcal{X}) \\ &= \sum_{A_j, \mathcal{Y}_j'} P^{a_i}(\mathcal{X}' | \mathcal{X}, A_j) P^{a_i}(\mathcal{Y}_i' | \mathcal{X}', A_j) P(A_j | M_j) \\ &\times P^{a_i}(M_j' | M_j, A_j, \mathcal{Y}_j', \mathcal{X}'). \end{aligned} \tag{3.1}$$

Here, the ADD $P^{a_i}(\mathcal{X}'|\mathcal{X}, A_j)$ compactly represents $T_i(s^{t-1}, a_i^{t-1}, a_j^{t-1}, s^t)$, $P^{a_i}(\mathcal{Y}_i'|\mathcal{X}', A_j)$ represents the probabilities $O_i(s^t, a_i^{t-1}, a_j^{t-1}, o_i^t)$, $P(A_j|M_j)$ represents $P(a_j^{t-1}|\theta_j^{t-1})$, and $P^{a_i}(M_j'|M_j, A_j, \mathcal{Y}_j', \mathcal{X}')$ represents the recursive belief update transitions $\tau_{\theta_j^t}(b_j^{t-1}, a_j^{t-1}, o_j^t, b_j^t) \times O_j(s^t, a_i^{t-1}, a_j^{t-1}, o_j^t)$ of the original I-POMDP.

3.1.3 Belief Update

Using the joint transition function computed above in equation 3.1, we can compute the I-POMDP$_\mathcal{X}$ belief update as:

$$b_i^{a_i,o_i}(\mathcal{X}', M_j') = \sum_{\mathcal{X}, M_j} b(\mathcal{X}, M_j) \times P^{a_i,o_i}(\mathcal{X}', M_j' | \mathcal{X}, M_j) \tag{3.2}$$

where $b(\mathcal{X}, M_j)$ represents the initial belief. The initial belief is also represented using ADDs as a joint distribution of $\mathcal{X}$ and M_j.

3.1.4 Solutions

For solving I-POMDP$_\mathcal{X}$ efficiently, we augment the IPBVI technique with ADDs for representating factors. Thus, we leverage the computaional advantages of IPBVI and the compactness of ADDs to make the I-POMDP$_\mathcal{X}$ tractable for larger problems.

We generalize the Symbolic Perseus solver [27] reviewed in section 2.1.4 for solving I-POMDP$_\mathcal{X}$.

The backup operation to generate α vectors using IPBVI is generalized for factored representations as follows,

$$\begin{aligned}
\Gamma^{a_i,*} \leftarrow \alpha^{a_i,*}(\mathcal{X}, M_j) &= \sum_{A_j} R^{a_i}(\mathcal{X}, A_j) P(A_j|M_j) \\
\Gamma^{a_i,o_i} \overset{\cup}{\leftarrow} \alpha^{a_i,o_i}(\mathcal{X}, M_j) &= \gamma \sum_{\mathcal{X}', M'_j} P^{a_i,o_i}(\mathcal{X}', M'_j|\mathcal{X}, M_j) \alpha^{t+1}(\mathcal{X}', M'_j), \quad \forall \alpha^{t+1} \in \mathcal{V}^{t+1} \\
\Gamma^{a_i} \leftarrow \Gamma^{a_i,*} \oplus_{o_i} \underset{\Gamma^{a_i,o_i}}{\arg\max}(\alpha^{a_i,o_i} \cdot b_i), &\quad \mathcal{V}^t \leftarrow \underset{\alpha^t \in \bigcup_{a_i} \Gamma^{a_i}}{\arg\max}(\alpha^t \cdot b_i), \quad \forall b_i \in B_i
\end{aligned} \tag{3.3}$$

Here, $\mathcal{V}^{t+1}$ is the set of α-vectors from the next time step and b_i is a belief point from the set of considered beliefs B_i. For populating B_i, we project the agent's beliefs H time steps into the future using equation 3.2. It is not practical to generate all possible beliefs from the initial belief b_0. This is because the branching factor for the belief tree is $|A_i| \times |O_i|$. So H time steps in the future, the belief tree will have a maximum of $\sum_{n=0}^{H}(|A_i| \times |O_i|)^n$ nodes. To avoid this exponential growth in the number of possible beliefs, future outcomes are simulated by sampling an action and an observation at each step, and using these to generate the next belief. We use stochastic simulation with greedy action (SSGA) strategy, also known as the ϵ-greedy exploration strategy to generate belief points [25]. In this method, an optimal action is chose with probability $1 - \epsilon$ and a random action is chosen with a small probability, ϵ. The optimal action is computed from the best available estimate of the value function at that time.

Several augmentations in the Symbolic Perseus solver such as cached ADD computations and ADD approximations further enable us to solve I-POMDP$_\mathcal{X}$ efficiently.

3.2 The cyber deception domain

In our work, we model the interaction between an attacker and a defender on a single honeypot host system. We describe the honeypot system and the attacker's mental states using a set of discrete

State Variable Name	Values	Description
PRIVS_DECEPTION	user, root, none	Deceptive reporting of privileges
S_DATA_DECOYS	yes, no	Presence of sensitive data decoys
C_DATA_DECOYS	yes, no	Presence of critical data decoys
HOST_HAS_DATA	sensitive_data, critical_data, none	Type of valuable data on the system
DATA_ACCESS_PRIVS	user, root	Privileges required to access or find data
ATTACKER_PRIVS	user, root	Attacker's highest privileges
DATA_FOUND	yes, no	Valuable data found by the attacker
VULN_FOUND	yes, no	Local *PrivEsc* discovered by attacker
IMPACT_CAUSED	yes, no	Attack successful
ATTACKER_STATUS	active, inactive	Presence of attacker on the host
HOST_HAS_VULN	yes, no	Presence of local *PrivEsc* vulnerability

Table 3.1: The state space of the cyber deception domain is comprised of 11 variables.

random variables. This section describes the state space, the attacker and defender's observation variables, actions, and reward functions.

3.2.1 States and observations

The state space of the cyber deception domain is made up of 11 state variables defining a total of 4,608 states. Table 3.1 briefly summarizes the state space. We broadly categorize the states into the following abstract categories:

- **States representing deception:** The PRIVS_DECEPTION, S_DATA_DECOYS, and C_DATA_DECOYS state variables belong to this category. The S_DATA_DECOYS and C_DATA_DECOYS state variables represent the presence of sensitive data decoys and critical data decoys. We differentiate between sensitive_data and critical_data as distinct targets. Sensitive data, for example, includes private data of employees, high ranking officials, or any data that

the attacker would profit from stealing. Critical data is type of data which is vital for the operation of a business process or a physical piece of equipment. The attacker would benefit from manipulating this type of data.

- **States describing the honeypot system:** These state variables represent the ground truth about the honeypot system. In our work, we focus on the presence or absence of various assets. The state variables belonging to this category are `HOST_HAS_DATA`, `DATA_ACCESS_PRIVS` and `HOST_HAS_VULN`. The `HOST_HAS_DATA` variable represents the true type of valuable data on the system. In practical scenarios, honeypots never contain any real valuable data. Hence, in our work, the `HOST_HAS_DATA` state is `none`. However, the attacker is unaware of the honeypot or the data decoys and hence can form a belief over this state variable. Thus, the `HOST_HAS_DATA` variable gives a subjective view of the attacker being deceived. As mentioned previously, we present `sensitive_data` and `critical_data` as distinct targets. We assume that a system cannot have two different types of valuable data simultaneously. This is a reasonable assumption because usually different hosts on enterprise networks possess different assets.

- **States describing the attacker's progress:** These state variables represent the attacker's privileges, presence and mental states about discovery of assets on the system. The state variables that belong to this category are `ATTACKER_PRIVS`, `ATTACKER_STATUS`, `DATA_FOUND`, `VULN_FOUND` and `IMPACT_CAUSED`. The `ATTACKER_PRIVS` state denotes the highest level of privileges available to the attacker. For the sake of simplicity, we assign two values to this variable; `user`, which represents an attacker with user level privileges, and `root`, an attacker with highest level of privileges. The defender cannot observe this state and has to formulate a strategy to keep attackers at both privilege levels engaged. The `ATTACKER_STATUS` state indicates if the attacker is still active on the honeypot. An attacker may decide to leave if the objective of the attack is accomplished or if there is no way of making progress. The

defender cannot observe this state and gets no information about it from any observations. This state has to be inferred from the defender's models of the attacker. DATA_FOUND and VULN_FOUND states indicate if the attacker has found data or local privilege escalation vulnerability.

Since some states are not completely observable, both agents get information about these states from their observations after performing actions. The attacker's observation space, $\mathcal{Y}_j$, contains 5 observation variables which make a total of 48 unique observations. These observation variables are: OP_RESULT, PRIVS, DATA, VULN, DISCREPANCY, and AOC_COMPLETE. The PRIVS, DATA, and VULN observations inform the attacker about the ATTACKER_PRIVS, DATA_FOUND and VULN_FOUND states. The OP_RESULT informs the attacker if the current action was completed without errors or failures. Besides, the attacker can also observe any anomalies through the DISCREPANCY observation variable. This is an abstract observation and has different interpretations for each attacker type. Broadly, if an attacker observes unexpected data or contradicting types of data, this observation is triggered. Thus the attacker's observations mainly serve the purpose of information gathering.

The defender's observations provide information about the attacker's activity and the attacker's interactions with any existing decoys. The defender's observation space, $\mathcal{Y}_i$, consists of 3 observation variables; S_DATA_DECOY_INTERACTION = { yes, no}, C_DATA_DECOY_INTERACTION = { yes, no}, and LOG_INFERENCE = { file_recon, sys_info, persist_attempt, none}. S_DATA_DECOY_INTERACTION and C_DATA_DECOY_INTERACTION observations alert the defender if the attacker interacts with sensitive_data decoys or critical_data decoys. The LOG_INFERENCE observation variable informs the defender about the attacker's actions inferred from log analysis. In our work, we use GrAAlf [29], a graph-based log analysis framework to implement the observation function for the LOG_INFERENCE variable. Inference from log analysis is often prone to noise from other background activity in the system. Hence this observation is noisy.

Action name	States affected	Description
FILE_RECON_SDATA	DATA_FOUND	Search for sensitive data for theft
FILE_RECON_CDATA	DATA_FOUND	Search for critical data for manipulation
VULN_RECON	VULN_FOUND	Search for local *PrivEsc* vulnerability
PRIV_ESC	ATTACKER_PRIVS	Exploit local *PrivEsc* vulnerability
CHECK_ROOT	none	Check availability of root privileges
START_EXFIL	IMPACT_CAUSED	Upload critical data over network
PERSIST	IMPACT_CAUSED	Establish a permanent presence in the system
MANIPULATE_DATA	IMPACT_CAUSED	Manipulate stored data
EXIT	ATTACKER_STATUS	Terminate the attack

Table 3.2: The actions available to the attacker.

3.2.2 Attacker and defender actions

At each step in the interaction, the attacker and defender can each act simultaneously. The attacker performs actions to gather information about the system, manipulate the system, or to take action on objectives. The defender's actions mainly govern the decoy deployment. Besides, the defender can also use instrumented *coreutils* programs to supply deceptive information to the attacker.

The actions available to the attacker, A_j, are briefly summarized in table 3.2. The FILE_RECON_SDATA and FILE_RECON_CDATA actions are used for data discovery. Depending on the action, the data discovery is successful if the attacker encounters sensitive_data or critical_data. VULN_RECON looks for exploitable privilege escalation vulnerabilities on a system. PRIV_ESC performs the privilege escalation exploit. This action usually leads to the attacker transitioning from user-level privileges to higher privileges. CHECK_ROOT is an information gathering action which informs the attacker about the highest available privileges. Depending on the type of the attacker, the START_EXFIL, MANIPULATE_DATA, or PERSIST actions can be performed to achieve

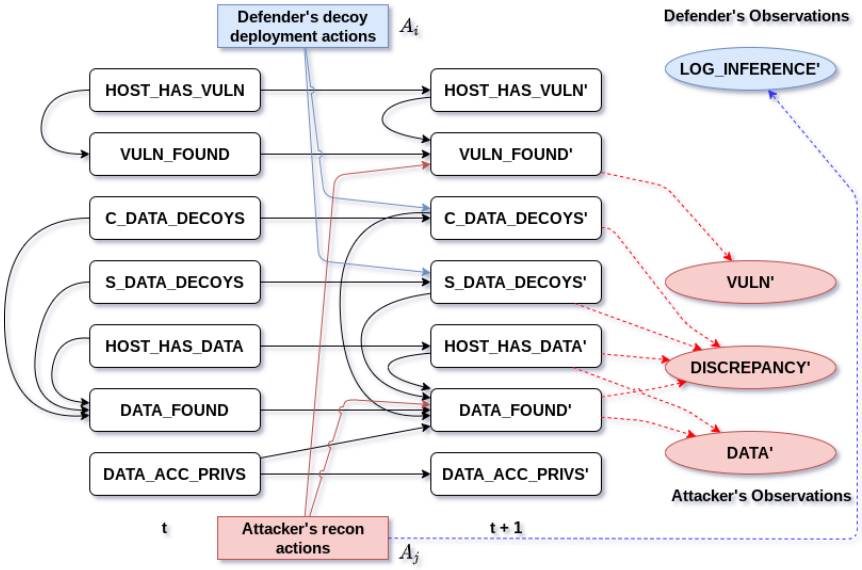

Figure 3.2: The dynamics for information-gathering actions compactly represented as a two time-slice DBN for select joint actions and observation variables.

the main objective.

The attacker can complete the attack only after locating the required assets and gaining the required privileges. Several different actions have to be performed sequentially to achieve this. Similarly, the defender has to accurately infer the intents of the attacker and deploy specific types of decoys to bait the attacker and learn more about their intent. The success or failure of these actions and the information gained by performing them is governed by the transition and observation functions. We consult cybersecurity experts to estimate the transition and observation probabilities in our work. Figure 3.2 shows the dynamics for information-gathering actions performed by the attacker. We now look briefly at how the attacker's and defender's actions affect the state space.

As previously mentioned, data discovery is an important part of the attacker's plan. The `FILE_RECON_SDATA` and `FILE_RECON_CDATA` actions, which are used for data discovery, cause the `DATA_FOUND` variable to transition to `yes`. `FILE_RECON_SDATA` action is slightly worse at finding data than the `FILE_RECON_CDATA`. This reflects the fact that private sensitive information is

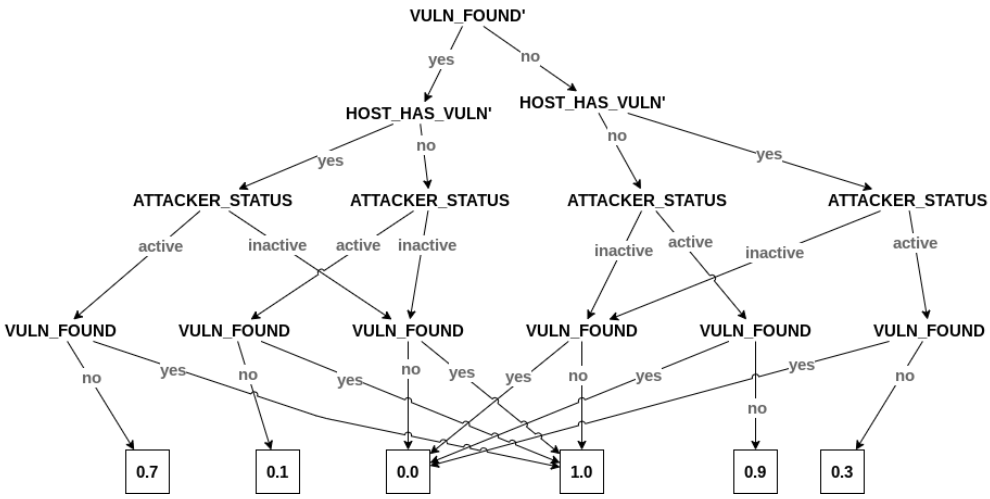

Figure 3.3: The ADD $P^{NOP}(\texttt{VULN\_FOUND}'|\mathcal{X}, A'_j = \texttt{VULN\_RECON})$. This ADD represents the joint transition function for the VULN_FOUND' state variable given $A_i = \texttt{NOP}$ and $A_j = \texttt{VULN\_RECON}$.

slightly difficult to find because it is often stored in user directories in arbitrary locations. On the other hand, critical data, like service configuration or database files, are stored in well-known locations on the system. We assume that the attacker is unable to discern between decoy data and real data, and hence, unable to determine which variable influences the DATA_FOUND state transition during file discovery. However, the attacker can become aware of contradicting or unexpected types of data if the wrong decoys are deployed. On observing DATA and DISCREPANCY observations simultaneously, the attacker develops a belief over the decoy data states as the host can have only one type of data. This realistically models a situation in which the attacker encounters multiple decoys of different types and suspects deception. VULN_RECON is another action that works similarly and causes the VULN_FOUND transition to yes. Figure 3.3 shows a part of this transition function in its ADD representation.

The defender starts with complete information about the system. Her actions mostly govern the deployment and removal of different types of decoys. These actions influence the S_DATA_DECOYS and C_DATA_DECOYS states. Besides baiting the attacker through decoys, the defender can influence

the attacker's observations about her privileges through the PRIVS_DECEPTION state. The set of actions available to the defender, A_i, is defined as $A_i = \{$ DEPLOY_S_DATA_DECOYS, DEPLOY_C_DATA_DECOYS, REMOVE_S_DATA_DECOYS, REMOVE_C_DATA_DECOYS, DEPLOY_VULN, SHOW_ROOT_PRIVS, NO-OP, SHOW_USER_PRIVS $\}$. The SHOW_USER_PRIVS and SHOW_ROOT_PRIVS actions use instrumented *coreutils* binaries to supply deceptive information about the attacker's privileges. These actions influence the PRIVS_DECEPTION state. In our work, we implement these actions on an adaptive honeypot running *Linux*. The deception is accomplished by showing lower or higher privileges regardless of the true privileges when the attacker uses commands like `whoami` or `id`. The DEPLOY_VULN action installs a vulnerable application or script on the system. The defender does this to facilitate privilege escalation for the attacker. This allows the defender to learn about the attacker's true motives.

We previously explained the noisy nature of the LOG_INFERENCE observation for the defender. Apart from the LOG_INFERENCE observation, the defender gets a near-perfect observation whenever the attacker interacts with a decoy through the S_DATA_DECOY_INTERACTION and C_DATA_DECOY_INTERACTION. Figure 3.4 shows the ADD for this observation function. The ADD for C_DATA_DECOY_INTERACTION is similar.

3.2.3 Rewards

We model three different attacker types in our work. These are, the *data exfil* attacker, the *data manipulator*, and the *persistent threat* type attacker. These attacker types are modeled as different frames in the defender's set of agent j's models, M_j. The reward function is different for each type of attacker.

The attacker is rewarded for exiting the system after causing an impact. However, the way this is achieved is different for each attacker type. For the *data exfil* attacker, performing the START_EXFIL action after finding `sensitive_data` accomplishes the attacker's objective. Similarly, for the *data manipulator*, performing MANIPULATE_DATA on `critical_data` accomplishes the attacker's

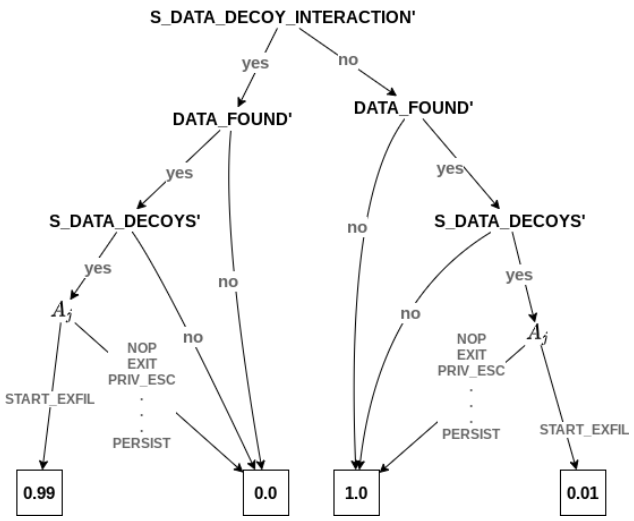

Figure 3.4: The ADD $P^{NOP}(\texttt{S\_DATA\_DECOY\_INTERACTION}'|\mathcal{X}', A'_j)$. This ADD represents the observation function for the `S_DATA_DECOY_INTERACTION` observation. This observation implies that the attacker performed the `START_EXFIL` action.

objective. The *persistent threat* attacker is rewarded for getting root level persistence in the system. Figure 3.5 shows the reduced ADD for the `EXIT` action for *data manipulator* attacker. The reward for the *data exfil* attacker is similar.

The defender's reward function is simpler. For every interaction step in which the `ATTACKER_STATUS` is `active`, the defender receives a small reward. Also, since I-POMDP reward functions are defined over joint actions, we apply a small cost for every $A_j = \texttt{EXIT}$ irrespective of A_i and state. This motivates the defender to prolong the interaction for as long as possible to delay the penalty from the attacker performing `EXIT`. Hence, the defender is not explicitly motivated to deceive the attacker. However, deception emerges as a behavior since the defender wants to delay the attacker's exit.

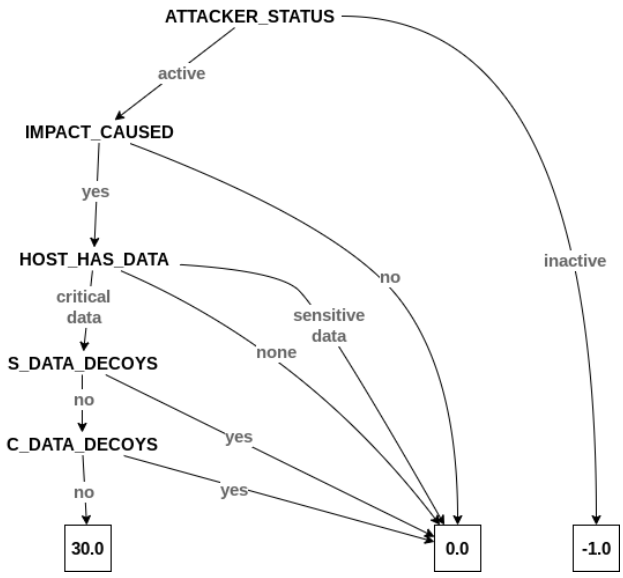

Figure 3.5: The reward ADD $\mathcal{R}^{EXIT}(\mathcal{X})$ for the *data manipulator* type attacker. The attacker is rewarded for performing the `EXIT` action if the `IMPACT_CAUSED` state is `yes` and `critical_data` is found.

3.3 Summary

We generalize finitely nested I-POMDPs to their factored form, I-POMDP$_\mathcal{X}$. ADDs are used to represent the dynamics of the I-POMDP$_\mathcal{X}$. Analogous to POMDPs, the belief update and point-based value backup for I-POMDPs can be extended to use factored representations. Due to the compact representations using ADDs and the fast approximate solutions enabled by IPBVI, I-POMDP$_\mathcal{X}$ proves to be tractable for larger problems. The I-POMDP$_\mathcal{X}$ framework is used to model a defender agent in a cyber deception interaction on a honeypot. The adaptive honeypot environment in which this interaction takes place is modeled using a set of state variables. The attacker acts to gather information, manipulate the system, and achieve specific objectives. The observations from the attacker's actions provide information about the presence of assets, vulnerabilities, and discrepancies in the system. The defender acts to manipulate the state of deception on the system. The defender's observations provide information about the attacker's interaction with deployed decoys, and the actions performed by the attacker. From these observations, the defender

infers the frame of the attacker. Multiple types of attackers are defined, each having different preferences and objectives. These attackers are modeled using their optimal level-0 POMDP policies.

Chapter 4

Attacker models

Targeted cyberattacks are implemented in multiple phases. Attackers usually start with user-level access inside the target organization. Starting from this initial phase, attackers have to maintain their access, elevate their privileges, and perform reconnaissance to locate their target before taking actions that cause impact. These phases are defined in various cyber-attack models such as the Lockheed Martin Cyber Kill Chain® [16], and the MITRE ATT&CK matrix [33]. The cyber deception domain described in the previous chapter is designed such that the attacker policies are analogous to these models to simulate realistic attackers.

We have previously defined the three distinct types of attackers which are modeled as separate frames in the I-POMDP. We implement each of these attackers using Metasploit [22], a well-known post-exploitation framework. To obtain the optimal policies for the attackers, we solve their level-0 POMDPs. Below we discuss the optimal policies for each attacker type.

4.1 The *data exfil* attacker frame

The *data exfil* type attacker is rewarded for stealing `sensitive_data` on the host. We model this attacker type based on threats that steal private data and other sensitive data from systems. The

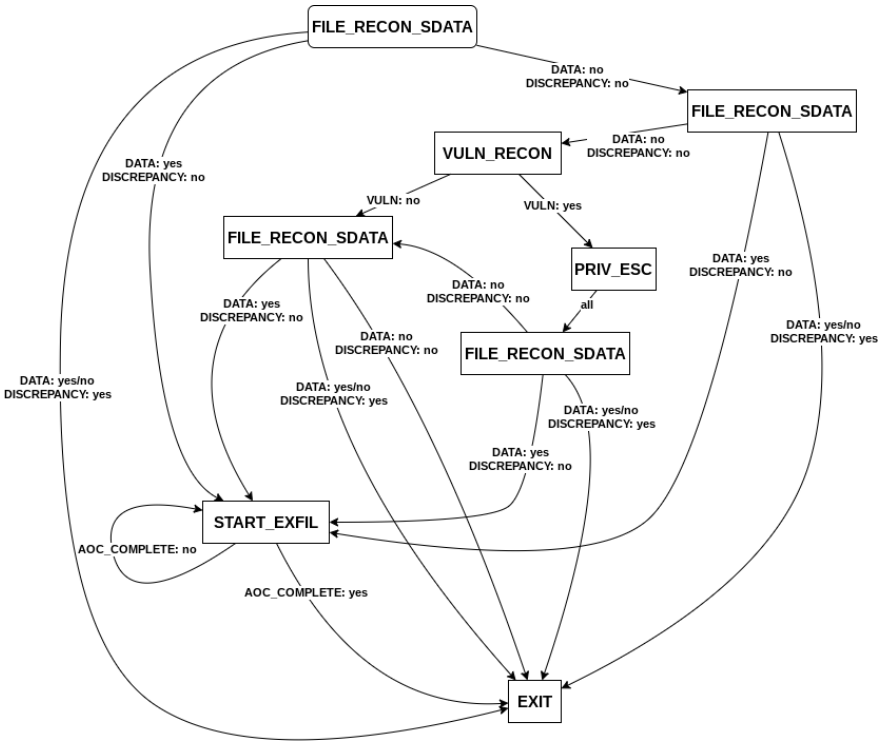

Figure 4.1: Optimal policy for *data exfil* type attacker. The *data exfil* attacker targets sensitive data on the system. The policy shows that the attacker starts with file discovery. On failure to find a file, the attacker escalates privileges and looks again before giving up.

attacker starts with no knowledge of the existence of data on the system. Figure 4.1 shows the optimal policy obtained by solving the level-0 POMDP. We see that the optimal policy recommends the `FILE_RECON_SDATA` action which simulates sensitive data discovery on computers. After failing to find data in the first few attempts, the attacker attempts to escalate privileges and search again. If the attacker encounters unexpected types of decoys, she leaves since there is no reward for stealing other types of data. Also, the observation of discrepancies when data is found informs the attacker about the possibility of deception. This is because the system only contains a single type of valuable asset. On being alerted to the possibility of being deceived, the attacker leaves the system.

4.2 The *data manipulator* attacker frame

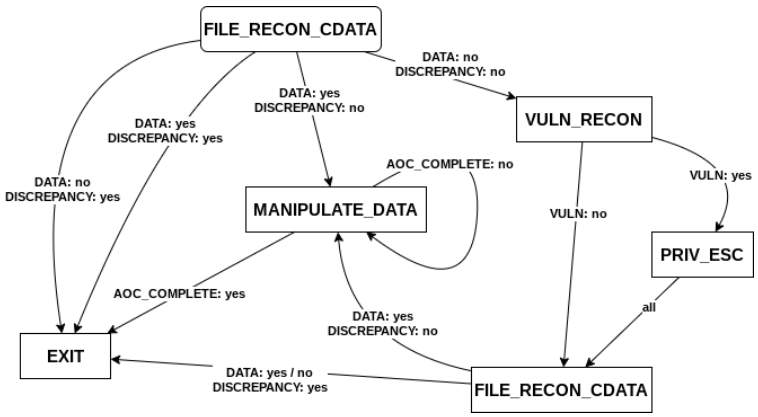

Figure 4.2: Optimal policy for *data manipulator* type attacker. The *data manipulator* attacker targets `critical_data` for manipulation.

The *data manipulator* type attacker is rewarded for manipulating `critical_data` on the host. Figure 4.2 shows the optimal policy for this attacker type. The attacker is modeled after adversaries that intrude systems to manipulate data that is critical for a business operation. Similar to the *data exfil* type, the attacker starts with no information about the existence of data. The optimal policy for this attacker type recommends `FILE_RECON_CDATA` action in the initial steps. Because critical data like service configurations or databases are usually stored in well-known locations, the `FILE_RECON_CDATA` is modeled to find `critical_data` quickly as compared to sensitive data. In the subsequent interaction steps, the attacker escalates privileges to continue the search if data is not found in the initial steps. Like the *data exfil* attacker, the *data manipulator* also leaves the system on observing discrepancies, suspecting deception, or on failure to find data.

4.3 The *persistent threat* attacker frame

The *persistent threat* aims to establish root level persistence on the host. Such attacks are common. Attackers establish a strong presence in an organization's network and stay dormant for an extended duration. Figure 4.3 shows the optimal policy for the *persistent threat*. For this attacker

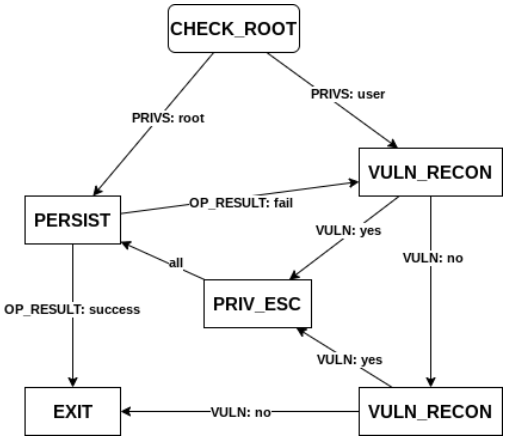

Figure 4.3: Optimal policy for *persistent threat* type attacker. Unlike the other attacker types, the *persistent threat* begins with vulnerability discovery actions. On finding a vulnerability and escalating privileges, the *persistent threat* attempts to establish a permanent presence in the system

type, the policy consists of vulnerability discovery actions in the initial steps. The attacker escalates privileges by performing the `PRIV_ESC` action on finding vulnerabilities. Once the attacker has the required privileges, the `PERSIST` action is performed to complete the objective.

While all three attacker policies may seem significantly different from their actions, the defender's observations of these actions are noisy. The errors in observation come from the noisy nature of real-time log analysis. For example, the `VULN_RECON` action models vulnerability discovery on a host. This action involves looking through the local file system for any vulnerable scripts, enumerating system information, listing services, etc. In real-time log analysis, a `VULN_RECON` can be mistaken for a `FILE_RECON_CDATA` or a `FILE_RECON_SDATA` which involve one or more similar steps. Similarly, it is difficult to tell the difference between the `FILE_RECON_CDATA` and `FILE_RECON_SDATA` from logs alone. Hence, without baiting the attacker into performing further actions, it is challenging to infer the intent of the attacker from the first few actions.

4.4 Summary

We model three different types of threats: the *data exfil* attacker, the *data manipulator* attacker, and the *persistent threat* through our attacker models. The *data exfil* attacker intends to steal sensitive data. *Data manipulator* intends to manipulate critical data. And the *persistent threat* aims to establish a strong presence at elevated privilege levels. To obtain the optimal policies for each of these attackers, we solve their level-0 POMDPS. These policies are based on the Lockheed Martin Cyber Kill Chain® and the MITRE ATT&CK matrix to make the models realistic. For the *data exfil* attacker, the optimal policy recommends data discovery actions in the initial steps. On failure to find data, the policy recommends privilege escalation. After getting elevated privileges, the attacker attempts to find data once again. On failure to find any data or on observing discrepancies, the attacker leaves the system. The *data manipulator* policy is very similar to the *data exfil* attacker. A key difference is that data discovery actions of the *data manipulator* are better than the *data exfil* attacker. Hence the attacker performs fewer data discovery steps before and after privilege escalation. Lastly, the policy for *persistent threat* skips the data discovery phase completely. The attacker starts with vulnerability discovery. After elevating privileges, the attacker performs persistence. We use this attacker models against the defender to evaluate its performance in realistic scenarios.

Chapter 5

Experiment results and discussion

To test the effectiveness of the I-POMDP$_\mathcal{X}$-based defender agent, we evaluate its performance on multiple scenarios in simulations and on an implemented adaptive honeypot. At each step in the interaction, we record the attacker's and defender's beliefs. We then compute the difference between these beliefs to determine if the defender was successful in recognizing the attacker's intent. In both sets of experiments, we compare the I-POMDP$_\mathcal{X}$-based defender against other passive defenders. These passive defenders simulate commonly used simpler deception deployment strategies. In this section, we describe the setup for these experiments and discuss the results.

5.1 Simulations

In simulations, we sample the state transitions and observations for both, the attacker and the defender from their respective observation functions. The frame and the starting privileges of the attacker are randomly sampled to simulate a threat with unknown intentions and privileges. The attacker starts with no prior knowledge about any vulnerabilities or data on the system.

As previously mentioned, we compare the I-POMDP$_\mathcal{X}$ policy against other passive baselines: one that does not engage and passively observes the attacker, and another which uses deception

Agent	Mean duration	Standard error	Max duration
I-POMDP$_\mathcal{X}$ agent	5.90	$\pm$ 0.24	9
NO-OP agent (no decoys)	4.30	$\pm$ 0.16	7
NO-OP agent (all decoys)	3.26	$\pm$ 0.20	5

Table 5.1: Interaction duration for each agent in simulations.

Agent	Cross-entropy mean	Cross-entropy standard error
I-POMDP$_\mathcal{X}$ agent	0.153	$\pm$ 0.080
NO-OP agent (no decoys)	0.649	$\pm$ 0.142
NO-OP agent (all decoys)	0.451	$\pm$ 0.101

Table 5.2: Mean cross-entropy at the final step after 30 simulations.

indiscriminately having deployed both sensitive and critical data decoys and all vulnerabilities in the honeypot at the beginning. We label the first baseline as NO-OP(no decoy) and the second baseline as NO-OP(all decoys). Each simulation trial lasts for 10 interaction steps. We perform 30 trials of the simulation. The I-POMDP$_\mathcal{X}$ policy converges in about 6 minutes with a mean time per backup of 37 secs on Ubuntu 18 with Intel i7 and 64 GB RAM. Table 5.1 shows the mean duration and max duration of 30 simulation trials.

The NO-OP(no decoy) and NO-OP(all decoy) yields a mean ($\pm$ std err.) of 4.3 $\pm$ 0.16 and 3.26 $\pm$ 0.20 steps of engagement with the attacker, respectively. The longest engagement among these last for 7 and 5 steps, respectively. With NO-OP(no decoy), the attacker is unable to find anything valuable on the system. Hence, after performing a few discovery actions and attempting to escalate privileges, the attacker exits. With NO-OP(all decoys), the agent either immediately finds the

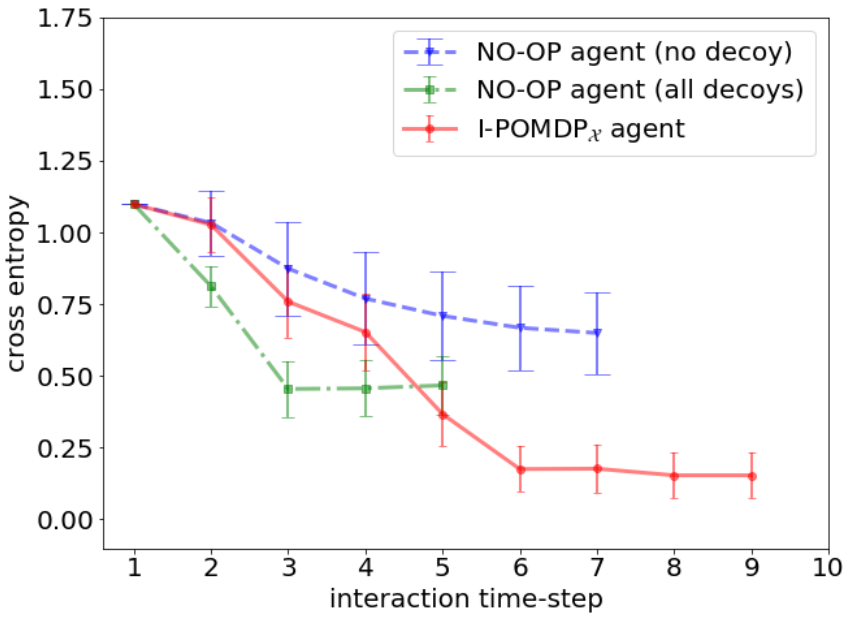

Figure 5.1: Mean cross-entropy between the defender's prediction of the attacker's frame and the attacker's true frame in simulations. The low cross entropy values for the I-POMDP$_x$ agent with low standard error shows that the I-POMDP$_x$ agent is significantly better at intent recognition.

target data or encounters unexpected decoys and suspects deception. In contrast, the I-POMDP$_x$ agent engages with the attacker for a mean duration of 5.9 ± 0.24 with the longest interaction happening for 9 steps. The I-POMDP$_x$-based defender leverages the information gained by the first few observations to selectively deploy the right type of decoys. In the initial steps, the defender performs the `SHOW_USER_PRIVS` action to manipulate the attacker's observations about her privileges. This increases her chances of observing file enumeration or vulnerability discovery activity, forming a belief over the frames. Subsequently, the defender baits the attacker using decoys and observes the interaction to solidify her belief. This minimizes the risk of the attacker encountering unexpected decoys or noticing discrepancies.

Figure 5.1 shows the cross-entropy between the defender's belief of the attacker's frame and

the attacker's true frame, as it varies across the steps of the interaction. The defender's I-POMDP$_\mathcal{X}$ policy eventually yields the lowest cross-entropy values compared to the baselines. In the initial steps, the noisy observations of the attacker's actions from log analysis help the defender form an approximate belief over the attacker's frame. For instance, multiple consecutive data reconnaissance steps filter out the persistence attacker type. However, observations from log inference are noisy. Hence, the defender cannot eliminate any of the attacker types. The cross-entropy decreases gradually as the interaction proceeds and more observations are made. After the initial few discovery actions, the final step of either exfiltrating the data or manipulating it allows the defender to distinguish between the two attacker types. For the *persistent threat* type attacker, the defender watches for vulnerability discovery actions. However, these actions also generate noisy observations for the defender. Hence, the defender facilitates privilege escalation and watches for possible persistence attempts. But, for the NO-OP(no decoy) with no deception, the only source of information about the attacker is from log inference. Hence, such a defender is unable to form accurate beliefs before the attacker leaves the interaction. For the NO-OP(all decoy) agent that indiscriminately uses deception, observations from decoy interactions are perfect, but the risk of the attacker encountering contradicting decoys and suspecting deception is also high leading to early exits. This is evident from the immediate sharp decrease in cross-entropy in the first few steps. But the high standard deviation suggests that this agent makes wrong predictions a significant number of times. Table 5.2 shows the mean cross-entropy between the defender's predictions and the truth at the final step of the simulation. By the final step, the I-POMDP$_\mathcal{X}$ agent has engaged with the attacker for enough duration to form very accurate beliefs about the frame of the attacker. In contrast, the other agents are not able to form accurate beliefs. This is evident from their higher mean cross-entropy and variance even in the final step.

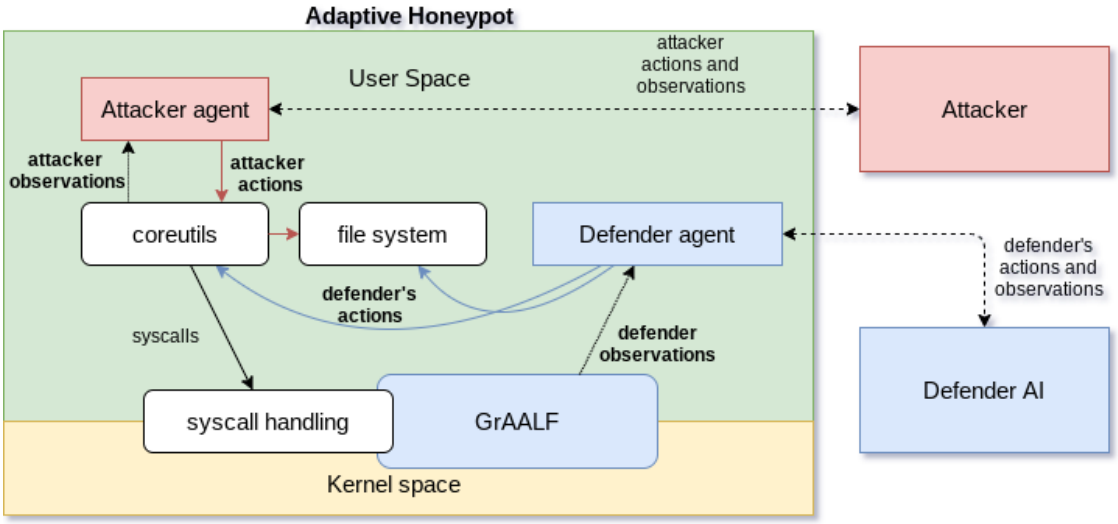

Figure 5.2: System architecture of the testbed used to deploy the agents. The defender manipulates the system through decoys and commonly used *coreutils* binaries to give deviant observations.

5.2 Host deployment

To implement the I-POMDP$_{\mathcal{X}}$ defender on a real system, we build an adaptive honeypot augmented with advanced logging abilities. Figure 5.2 shows the overall architecture of our testbed implementation. The testbed consists of 3 separate hosts: the *attacker*, the *adaptive honeypot* and the *defender*. The *attacker* system operates a Kali Linux distribution which is well known for the variety of offensive and defensive cybersecurity tools that are preinstalled on it. The *adaptive honeypot* on which the interaction takes place is a Metasploitable 3 Linux distribution. This distribution has a wide range of builtin vulnerabilities and is commonly used to simulate victim workstations in cyber-attack simulations. The *adaptive honeypot* also contains an *attacker agent* which executes the actions given by the attacker. We program the *attacker agent* to use realistic techniques commonly used by real attackers. For instance, privilege escalation is done by exploiting real vulnerabilities on the host. The *adaptive honeypot* also has a defender agent that implements the defender's actions and gets observations.

Agent	Mean duration	Standard error	Max duration
I-POMDP$_{\mathcal{X}}$ agent	5.90	$\pm$ 0.28	10
NO-OP agent (no decoys)	4.44	$\pm$ 0.11	5
NO-OP agent (all decoys)	3.86	$\pm$ 0.23	6

Table 5.3: Interaction duration for each agent on the adaptive honeypot implementation.

Agent	Cross-entropy mean	Cross-entropy standard error
I-POMDP$_{\mathcal{X}}$ agent	0.068	$\pm$ 0.030
NO-OP agent (no decoys)	0.410	$\pm$ 0.139
NO-OP agent (all decoys)	0.256	$\pm$ 0.073

Table 5.4: Mean cross-entropy at the final step after 30 testbed runs.

The defender AI located on the *defender* workstation recommends the optimal action for each interaction step by solving the I-POMDP$_{\mathcal{X}}$. To implement the observation function for the `LOG_INFERENCE` observation variable, we use GrAALF [29], a graphical framework for processing and querying system call logs. GrAALF analyzes system call logs in real-time and provides the stochastic `LOG_INFERENCE` observation variable values for file and vulnerability searches. Besides, GrALLF also provides the perfectly observed `DATA_DECOY_INTERACTION` variable values to the defender.

Table 5.3 shows the interaction duration for 30 trials on the testbed. The duration of the engagements for the I-POMDP$_{\mathcal{X}}$ is consistent with the results of the simulation with minor differences. Figure 5.3 shows the cross-entropy between the frame predicted by the defender and the attacker's actual frame. The cross-entropy values are consistent with the simulations for the I-POMDP$_{\mathcal{X}}$

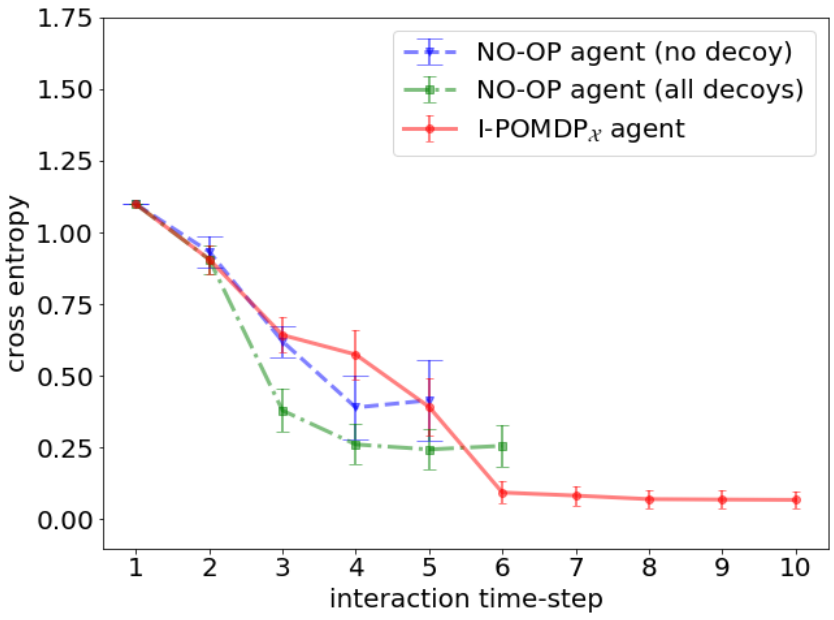

Figure 5.3: The cross-entropy values for the I-POMDP$_\mathcal{X}$ agent on the testbed runs are consistent with the results in simulations.

agent and the NO-OP agent that does not use deception. However, the NO-OP agent that uses all decoys seems to perform better than in simulations. However, the high standard error implies that this strategy has a very high rate of failure. Despite this difference, the I-POMDP$_\mathcal{X}$ agent significantly outperforms the other two at intent recognition. The performance of all three agents at intent recognition is shown in table 5.4.

5.3 Instances of active deception

We record the beliefs of both agents in our experiments as the interaction proceeds. Below, we show a few instances of the defender using deception to influence the attacker's beliefs.

Figure 5.4 illustrates a scenario taken from an actual simulation run with the *data manipulator*

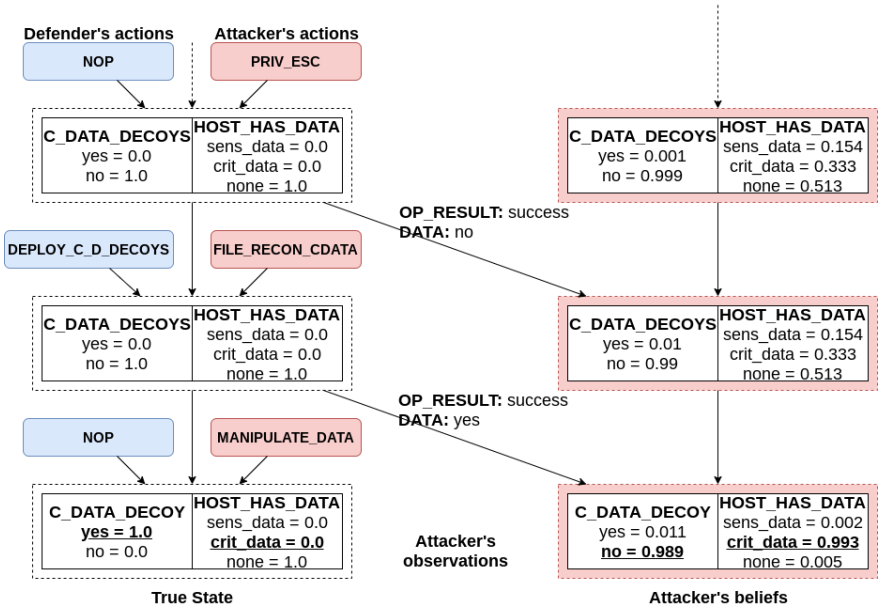

Figure 5.4: The attacker starts with a low prior belief on the existence of decoys and an active defender. If decoys are indistinguishable from real data, the attacker attributes his observation to the existence of real data even when the host has none.

attacker type. Initially, the attacker has a non-zero belief over the existence of data on the system. However, the true state of the honeypot on the left shows that it does not contain any data. In the absence of the defender or any static data decoys, the attacker will eventually realize this after `FILE_RECON_CDATA` fails to find any data. However, before this happens, the defender deploys data decoys when the attacker acts. The attacker's inability to tell the difference between decoy data and real data and his prior belief about the absence of decoys leads her to attribute her observations to the existence of real data. This is evident from the attacker's beliefs which show a high probability for the `critical_data` value of the `HOST_HAS_DATA` state variable.

Figure 5.5 shows another scenario taken from simulations. In this particular scenario, the defender observed a file discovery action in the beginning and deployed critical data decoys. However, subsequent observations made by the defender were inconsistent with the *data manipulator* type attacker. Hence, the defender switches the decoys before the attacker can spot any discrepan-

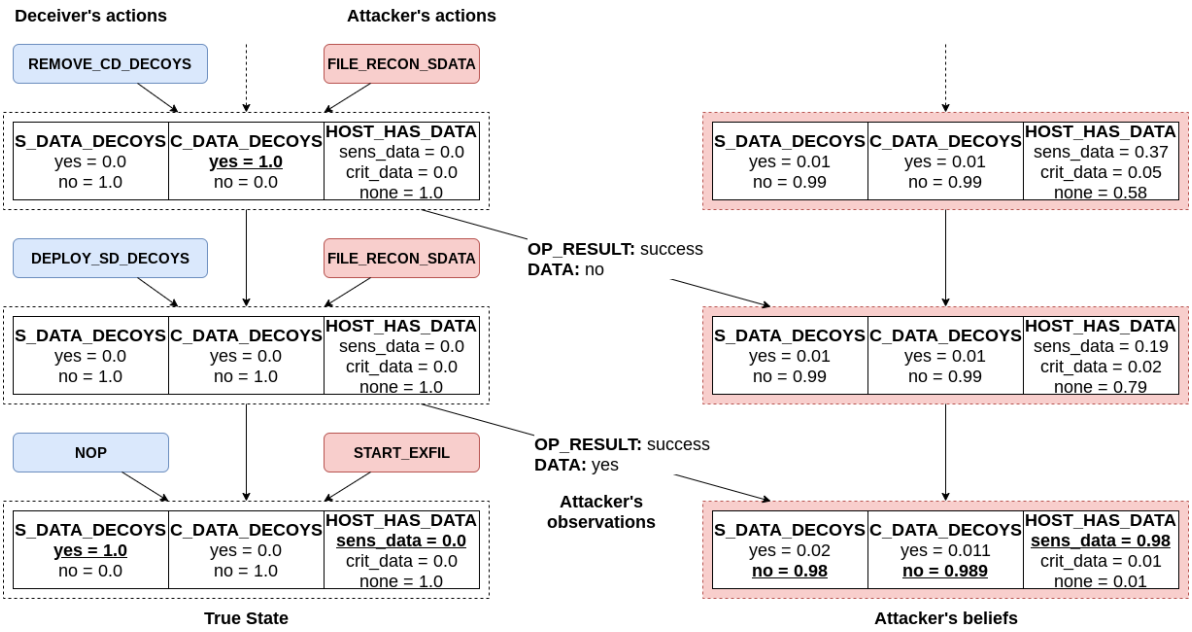

Figure 5.5: In the event of deploying wrong decoys, the defender corrects the decoy deployment. In this case, on observing file discovery actions, the defender deployed critical data decoys. Later as the interaction progresses, the defender forms a better belief over the attacker's frame from the observation and replaces the decoys before the attacker discovers the discrepancy.

cies. The true state of the system is shown on the left. The defender performs `REMOVE_CDATA_DECOYS` when the `C_DATA_DECOYS` state is `yes`. Simultaneously, the attacker performs `FILE_RECON_SDATA`. In such a scenario, the defender's action is given priority. Hence the attacker is unable to find data and the next state shows that `C_DATA_DECOYS` has transitioned to `no`. The attacker is unable to find any data and has a stronger belief that the host might not have any data. As the attacker performs the `FILE_RECON_SDATA` action for the last time, the defender deploys sensitive data decoys. In this particular case, the attacker was able to find the decoys and interact with them. In some cases, the attacker is unable to find the decoys despite the defender deploying them due to the imperfect nature of the `FILE_RECON_SDATA` action. When this happens, the defender does not observe any decoy interactions and is unable to form an accurate belief over the frame of the attacker.

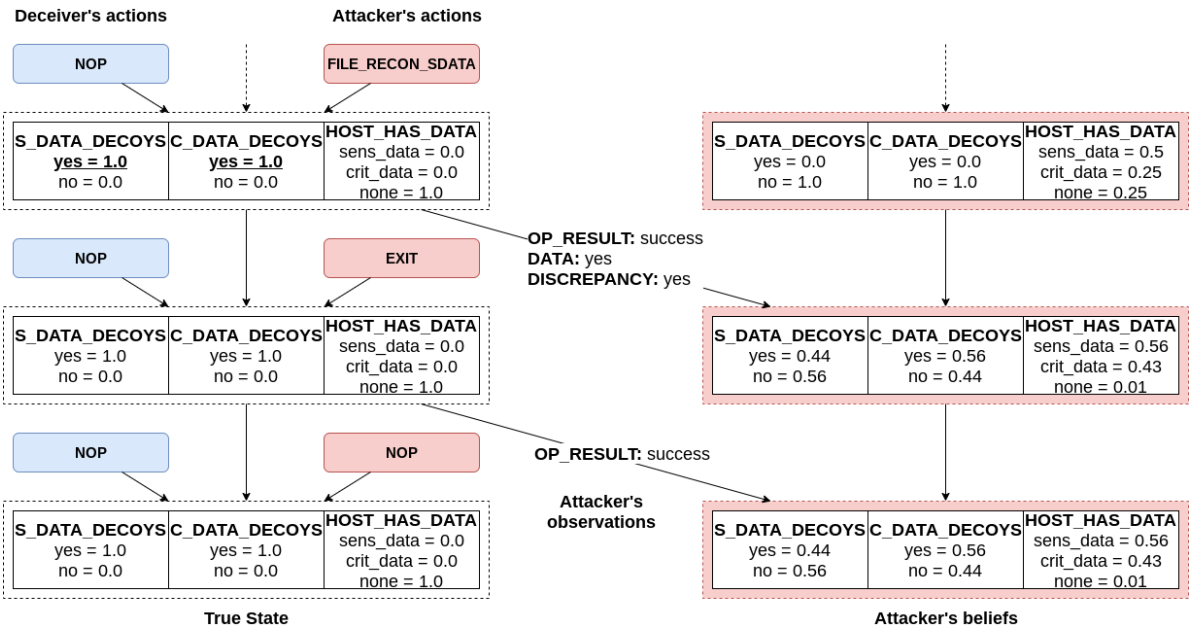

Figure 5.6: The attacker suspects deception on observing discrepancies. On performing `FILE_RECON_SDATA`, the attacker gets the `DATA` and the `DISCREPANCY` observation simultaneously. This leads to non-zero beliefs over the existence of decoys and the attacker exits in the next step.

The next scenario is taken from a simulation with the NO-OP (all decoys) agent. The beliefs of the attacker as the simulation proceeds are shown in figure 5.6 on the right. For the NO-OP agent, all decoys are deployed in the beginning. The figure shows this through the perfect probabilities of the `S_DATA_DECOYS` and `C_DATA_DECOYS` states on the left. On performing the `FILE_RECON_SDATA` action, the attacker encounters both decoys. But the `HOST_HAS_DATA` state implies that the system can only contain a single type of asset. Hence, the attacker forms a belief over the decoys states. This leads the attacker to promptly exit the system.

The scenarios mentioned above demonstrate how a defender can actively engage with the attacker using deception. In our work, the I-POMDP$_{\mathcal{X}}$-based defender is able to leverage the imperfect information gained from log inference to formulate better deception strategies. This shows the value of AI-based active deception in engaging with attackers.

5.4 Summary

We implement the attacker agents discussed in chapter 4 and the I-POMDP$_{\mathcal{X}}$-based defender agent on a real system. We evaluate the I-POMDP$_{\mathcal{X}}$-based defender against a randomly sampled attacker with randomly assigned privileges. We compare the I-POMDP$_{\mathcal{X}}$-based defender against two other simpler agents that simulate commonly used deception strategies. Our results show that the I-POMDP$_{\mathcal{X}}$-based defender agent can engage with the attacker significantly longer than the other agents. Further, at the end of the interaction, on an average the I-POMDP$_{\mathcal{X}}$-based agent has the lowest cross-entropy values between the predicted frame and the true frame. This shows that the I-POMDP$_{\mathcal{X}}$ agent outperforms the other agents at intent recognition. We perform all these experiments and evaluations in simulations and on a real honeypot which we implement. Our results are consistent in both cases. This shows that I-POMDP$_{\mathcal{X}}$ can be effective for intent recognition in practical scenarios. We also illustrate how deception affects the attacker's beliefs using select instances of the interactions taken from actual simulations

Chapter 6

Conclusion

Cyber deception is being rapidly adopted as an effective strategy for cyber defense. Deception increases the uncertainty for the attackers and improves the defenders' reaction time. Despite its numerous advantages, however, deception is mostly used to detect and/or confuse the attackers. Our work demonstrates that deception can be used dynamically to actively engage threats and learn more about their intent. Intent recognition yields more information about the enemy and is vital in mounting an effective cyber defense.

Our approach of utilizing automated decision-making for deception to recognize attacker intent is a novel application of AI and decision-making in cybersecurity. Broadly, we show that cyber deception for intent recognition can be formulated as a multi-agent automated decision-making problem. To compute optimal deception strategies for this problem, we introduce a factored variant of the I-POMDP framework. Our experiments reveal that the I-POMDP$_\mathcal{X}$-based agent succeeds in engaging various types of attackers for a longer duration than passive honeypot strategies. The results from the interactions between the implemented attacker and defender agents show that longer interactions indeed facilitate intent recognition. Importantly, the I-POMDP$_\mathcal{X}$-based agent can be practically implemented on a real system with logging capabilities paving the way for its deployment in actual honeypots.

6.1 Limitations

While the I-POMDP$_\mathcal{X}$ is effective at solving larger problems, its advantages mainly depend on the level of abstraction of the state space and observation space. In scenarios involving cybersecurity, the computational complexity of I-POMDP$_\mathcal{X}$ solutions can often be a limiting factor. Active cyber defense requires fast and efficient online policies.

Also, in our work, we assume that the attackers are rational agents. This may be a strong assumption since attackers are humans and can act irrationally. While it is possible to account for irrational behavior by assigning a small probability ϵ to $P(A_j|M_j), \forall a_j \notin OPT(A_j)$, it greatly increases the number of opponent's models in the interactive state space. This further increases the time required for computing the optimal policy.

6.2 Future work

There are several aspects of our work that can be improved upon. An obvious area to improve will be to build a faster solver for the I-POMDP$_\mathcal{X}$ and scale it to very large domains. This will enable better solutions to the I-POMDP$_\mathcal{X}$ within realistic time bounds. Additionally, the proposed cyber deception domain can be extended to multi-host settings. This will enable the defender to model other key phases of cyberattacks such as lateral movement, network discovery, etc. Modeling multiple hosts also opens up the possibility of using network-level deception. This will facilitate longer and realistic engagements which can reveal much more about the attacker. Lastly, the nesting level of the I-POMDP$_\mathcal{X}$ can be increased to model possible counter deception strategies that the attackers may use. As deception becomes increasingly popular, modeling deception-aware attackers may reveal how attackers might counter commonly used deception strategies.

We believe that our work will highlight the interesting capabilities that AI has to offer in the area of automated cyber defense.

References

[1] Karl Johan Åström. Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, 1965.

[2] R Iris Bahar, Erica A Frohm, Charles M Gaona, Gary D Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebric decision diagrams and their applications. *Formal methods in system design*, 10(2-3):171–206, 1997.

[3] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in bayesian networks. *arXiv preprint arXiv:1302.3562*, 2013.

[4] Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1168–1175. Citeseer, 1996.

[5] Thomas E. Carroll and Daniel Grosu. A game theoretic investigation of deception in network security. *Security and Communication Networks*, 4(10):1162–1172, 2011.

[6] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational intelligence*, 5(2):142–150, 1989.

[7] Daniel Dennett. Intentional systems. brainstorms, 1986.

[8] Prashant Doshi. Decision making in complex multiagent settings: A tale of two frameworks. *AI Magazine*, 33(4):82–95, 2012.

[9] Prashant Doshi and Dennis Perez. Generalized point based value iteration for interactive pomdps. In *AAAI*, pages 63–68, 2008.

[10] Prashant J Doshi. *Optimal sequential planning in partially observable multiagent settings*. PhD thesis, 2005.

[11] Karel Durkota, Viliam Lisỳ, Branislav Bošanskỳ, and Christopher Kiekintveld. Approximate solutions for attack graph games with imperfect information. In *International Conference on Decision and Game Theory for Security*, pages 228–249. Springer, 2015.

[12] Zhengzhu Feng and Eric A Hansen. Approximate planning for factored pomdps. In *Sixth European Conference on Planning*, 2014.

[13] Kimberly Ferguson-Walter, Sunny Fugate, Justin Mauger, and Maxine Major. Game theory for adaptive defensive cyber deception. In *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, page 4. ACM, 2019.

[14] Piotr J Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.

[15] Eric A Hansen and Zhengzhu Feng. Dynamic programming for pomdps using a factored state representation. In *AIPS*, pages 130–139, 2000.

[16] Eric Hutchins, Michael Cloppert, and Rohan Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *6th International Conference on Information Warfare and Security, ICIW 2011*, (July 2005):113–125, 2011.

[17] Sushil Jajodia, Noseong Park, Fabio Pierazzi, Andrea Pugliese, Edoardo Serra, Gerardo I Simari, and VS Subrahmanian. A probabilistic logic of cyber deception. *IEEE Transactions on Information Forensics and Security*, 12(11):2532–2544, 2017.

[18] Sushil Jajodia, VS Subrahmanian, Vipin Swarup, and Cliff Wang. *Cyber deception*. Springer, 2016.

[19] AH Jesse Hoey, Robert St Aubin, and Craig Boutilier. Spudd: stochastic planning using decision diagrams. *Proceedings of Uncertainty in Artificial Intelligence (UAI). Stockholm, Sweden. Page (s)*, 15, 1999.

[20] Anthony R Kaelbling, Leslie Pack and Littman, Michael L and Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101:99—-134, 1998.

[21] Nicholas S Kovach, Alan S Gibson, and Gary B Lamont. Hypergame theory: a model for conflict, misperception, and deception. *Game Theory*, 2015, 2015.

[22] David Maynor. *Metasploit toolkit for penetration testing, exploit development, and vulnerability research*. Elsevier, 2011.

[23] George E Monahan. State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms. *Management science*, 28(1):1–16, 1982.

[24] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[25] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Anytime point-based approximations for large pomdps. *Journal of Artificial Intelligence Research*, 27:335–380, 2006.

[26] Pingree. Emerging Technology Analysis : Deception Techniques and Technologies Create Security Technology Business Opportunities. *Trapx security*, pages 1–18, 2018.

[27] Pascal Poupart. *Exploiting structure to efficiently solve large scale partially observable Markov decision processes*. PhD thesis, University of Toronto, 2005.

[28] Aaron Schlenker, Omkar Thakoor, Haifeng Xu, Long Tran-Thanh, Fei Fang, Phebe Vayanos, Milind Tambe, and Yevgeniy Vorobeychik. Deceiving cyber adversaries: A game theoretic approach. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2:892–900, 2018.

[29] Omid Setayeshfar, Christian Adkins, Matthew Jones, Kyu Hyung Lee, and Prashant Doshi. Graalf: Supporting graphical analysis of audit logs for forensics. *arXiv preprint arXiv:1909.00902*, 2019.

[30] M. Simaan and Jr. J.B. Cruz. On the stackelberg strategy in nonzero-sum games. *Journal of Optimization Theory and Applications*, 11(5):533–555, 1973.

[31] EJ Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, 1971.

[32] Lance Spitzner. The honeynet project: Trapping the hackers. *IEEE Security & Privacy*, 1(2):15–23, 2003.

[33] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. Mitre att&ck: Design and philosophy. *Technical report*, 2018.

[34] Joseph A Tatman and Ross D Shachter. Dynamic programming and influence diagrams. *IEEE transactions on systems, man, and cybernetics*, 20(2):365–379, 1990.

[35] Nikos Vlassis, Matthijs TJ Spaan, et al. A fast point-based algorithm for pomdps. In *Benelearn 2004: Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands*, pages 170–176, 2004.