A Markov Decision Process Model for High Interaction Honeypots

Osama Hayatle[1], Hadi Otrok[2], Amr Youssef[1]
[1]Concordia Institute for Information Systems Engineering
Concordia University, Montreal, Quebec, Canada
[2] Electrical and Computer Engineering Department
Khalifa University of Science, Technology & Research, Abu Dhabi, UAE
Email:{youssef, o_hayat}@ciise.concordia.ca, Hadi.Otrok@kustar.ac.ae

Abstract

Honeypots, which are traps designed to resemble easy to compromise computer systems, have become essential tools for security professionals and researchers because of their significant contribution in disclosing the underworld of cybercirmes. However, recent years have witnessed the development of several anti-honeypot technologies. Botmasters can exploit the fact that honeypots should not participate in illegal actions by commanding the compromised machine to act maliciously against specific targets which are used as sensors to measure the execution of these commands. A machine that is not allowing the execution of such attacks is more likely to be a honeypot. Consequently, honeypots operators need to choose the optimal response that balances between being disclosed and being liable for participating in illicit actions. In this paper, we consider the optimal response strategy for honeypot operators. In particular, we model the interaction between botmasters and honeypots by a Markov Decision Process (MDP) and then determine the honeypots optimal policy for responding to the commands of botmasters. The model is then extended using a Partially Observable Markov Decision Process (POMDP) which allows operators of honeypots to model the uncertainty of the honeypot state as determined by botmasters. The analysis of our model confirms that exploiting the honeypots legal liability allows botmasters to have the upper hand in their conflict with honeypots. Despite this deficiency in current honeypot designs, our model can help operators of honeypots to determine the optimal strategy for responding to botmasters' commands. We also provide simulation results that show the honeypots optimal response strategies and their expected rewards under different attack scenarios.

Index Terms

Honeypots, Botnets, Markov Decision Process

I. Introduction

Through the long combat between security professionals and attackers, the latter were always one step ahead. It has always been the case that attackers create new hacking tools, use them for a while, and then security professionals come up with solutions against these tools. However, the game rules may change with the wide use of honeypots. Honeypots are traps designed to resemble easy-to-compromise systems in order to tempt hackers to invade them and then collect valuable information about the botmasters' techniques, tools and even their true identities. When attackers target a honeypot, all their commands, techniques and codes are captured as they become under the observation of security professionals who operate the honeypot. Depending on their complexity, honeypots are categorized into two groups (i) high interaction honeypots that resemble real machines with diversity of services and (ii) low interaction honeypots that resemble machines with limited services [1].

Honeypots are becoming a major source of information for security communities and are used around the world to capture and analyze information about attackers. On the other hand, detecting honeypots has become an active area of research by both attackers and security professionals [2], [3]. In [4], Zou and Cunningham suggested a detection methodology by exploiting the legal liability of honeypots when participating in illegal actions such as Distributed Denial of Service (DDoS) or malware spreading. In this methodology, botmasters command the compromised machine to attack targets that act as sensors. Then, the machine response to these commands is measured by the sensors and its true nature is determined. To prevent honeypots from recognizing these tests, Zou and Cunningham suggested to mix test commands with real attack commands such that the honeypot operators may be legally liable if decided to execute botmaster's commands. It is not possible for current honeypot technology to avoid such detection technique. *This raises the need for deciding the optimal strategy that prolongs honeypots stay in the botnet while avoiding high legal liability.* In this paper, we model the interaction between the honeypot and the botmaster using Markov Decision Process (MDP) [5] in order to determine the honeypot optimal response to such test techniques. In particular, in the first part of this work, the honeypot-botmaster interaction is modeled by MDP with a set of states, actions, and transition probabilities. Depending on the beginning and end states, the system may acquire rewards or costs in each transition. These transitions are determined by the current state, the selected action, and the transitions probabilities. Our work shows how honeypot operators can select the optimal strategy by considering different factors and parameters, e.g., liability cost, honeypot operation cost, honeypot reset cost, probability of attacks and probability of disclosure. Then we extend our model to a more realistic scenario using Partially Observable Markov Decision Process (POMDP) which allow us to deal with

the uncertainty associated with the fact that the honeypot state as determined by the botmaster is not known to the honeybot operator.

The models developed in this paper can help security professionals to:

- Determine the optimal responses to botmasters commands.
- Prolong the honeypot lifetime inside botnets while minimizing its legal liability.

The remaining of this paper is organized as follows. In section II, we briefly review some related work in the area of honeypot detection and the use of MDP in the area of information systems security. In section III, we describe our MDP-based model. In section IV, we discuss how to select the optimal policy for our model based on available parameters. In section V, we extend the developed model using POMDP. Our simulation results and discussion are presented in section VI. Finally we summarize our conclusions in section VII.

II. Related Work

In this section, we briefly review some of the research related to the area of honeypots detection and the use of Markov Decision Processes in information systems security.

A. Honeypots Detection

The wide use of honeypots as security surveillance sensors motivated hackers, as well as security researchers, to explore the weaknesses of these important tools and develop techniques and tools that can be used to disclose them. Ferrie [2] described techniques that can be used to detect virtual machines which are usually used to deploy honeypots. Detecting virtual environments relies on measuring the latencies that take place when communicating between the host operating system and the virtual machine. By comparing these latencies with the measurements of other normal machines, virtual machines can be recognized. The common deployment of honeypots as virtual machines encouraged researchers to investigate other methodologies of virtual machines detection. In [6], Fu *et al.* showed the possibility to detect virtual environments by measuring the network connections time in suspicious machines and comparing them to other machines in the same environment. Krawtez [3] showed how the 'Honeypot Hunter' tool is able to detect honeypots by determining their ability to send spam emails. The operators of honeypots do not allow spam emails to be sent from within honeypots. Thus, by commanding the honeypot to send spam emails, that are received by the Honeypot Hunter, it is possible to distinguish honeypots from real machines. Hayatle *et al.* [7] used Dempster-Shafer theory to combine different evidence and calculate the belief about the true nature of the machine. In particular, the authors in [7] suggested to collect evidence throughout different phases of machine compromise, to assign a belief value for each evidence that reflects the evidence support for the machine type, i.e., normal or honeypot, and then combine the beliefs using Dempster-Shafer theory. This approach enables attackers to systematically collect and combine evidence while interacting with a suspicious machine to make a decision about its true nature.

B. Using Markov Decision Process in Security

Markov Decision Processes are widely used as optimization tools for determining optimal strategies in automated systems, e.g., see [8], [9]. Jha *et al.* [10] explained how to interpret attack graphs as MDP models and solve these models to determine the optimal defense strategy that minimizes the probability of attack success. Taibah *et al.* [11] presented a dynamic defense architecture against email worms where they classified emails into categories based on their threat level, and then used a MDP model to select the optimum set of actions, e.g., quarantine, analyze, or drop, that can be applied to maximize the architecture outcome by increasing security and decreasing the latency. The work in [12] employs MDP to determine the optimal action against attack attempts where the author found that depending on attack severity, it is not always optimal to defend against attacks as it is possible that the overhead caused by continuous defense strategy may exceed the cost of the attack itself. The findings were analyzed in the cases of perfect and imperfect detectors using MDP and POMDP. Liu *et al.* [13] used POMDP to design a framework for combining intrusion detection and continuous authentication in mobile ad hoc networks (MANETs).

III. Markov Decision Process Model for High Interaction Honeypot

In this section, we briefly review the principles of Markov Decision Processes (MDP), and present our developed MDP model for honeypot interaction with botmasters. A Markov Decision Process [14] is a tuple (S, A, P, r, d), where S represents the set of system states, A represents the set of possible actions, and *P* is a transition function

$$P : S \times S \times A \longmapsto [0, 1]$$

where $P(s_1, s_2, a)$ is the probability of transiting from state s_1 to state s_2 upon using action a. The reward function r represents the gain for using action a in state s:

$$r : S \times A \longmapsto \mathbb{R}$$

The discount factor d is the amount by which the gain is discounted over time.

A. Honeypot States

At any given time, the honeypot can be in one of three possible states:

1) W (Waiting state): The honeypot is not targeted by the attacker yet and is waiting for attack attempts so that it can join a botnet. In this state, the honeypot cannot capture any information about the attackers as it does not have any interaction with them yet.
2) C (Compromised state): The honeypot has been successfully compromised by attackers and has become a member of their botnet. In this state, the honeypot is able to collect information about the attackers whenever they communicate with it. It is worth mentioning that some information are gained during the compromising phase, i.e., during the transition between state W and state C.
3) D (Disclosed state): The true nature of the honeypot is detected by the attackers and it is no longer a member of their botnet. This can be the case when receiving a command to disconnect or remove the honeypot from the botnet. It is also possible to think that a honeypot is disclosed when losing the interaction with the botmasters for a relatively long period. However, security professionals must consider different phases of the botnet life cycle [15] as it is possible that botmasters focus on expanding their botnets and do not communicate intensively with existing botnet members.

B. States Transitions

The transition from one state to another is determined by different factors and is associated with different rewards. In our model, transitions between states depend on two factors:

1) Honeypot Actions: At each state, the honeypot can choose one of the following three Actions:
   - A (Allow) the honeypot allows the botmasters to compromise the system and to execute commands such as downloading files, installing and updating the bot code, and launching attacks from within the system. This enables the honeypot to infiltrate the botnet and prolong its stay in it. However, it comes with the cost of possible liability in case of participating in real attacks.
   - N (Not-Allow): If a honeypot chooses this action at state W, then the honeypot will not let the attackers compromise it. Consequently, the honeypot will not be able to join any botnet. After compromising the system, i.e., system is in state C, this action is used to prevent the attackers from sending malicious traffic from the honeypot. One reason for following this action is to avoid the liability of participating in illegal actions. Also the honeypot can be designed to reject some commands from the attackers in order to force them to use new tools and techniques [16]. However, ignoring the botmasters' commands may allow them to disclose the honeypot true nature.
   - R (Reset): the honeypot is reset to its initial state (W). The honeypots cannot collect information after being disclosed by the attacker. Thus, to make use of these resources, honeypots must be reinitialized as new honeypots and be ready to lure new attackers.
2) Transition Probabilities: Beside the actions discussed above, the transitions between the different states in the model are also determined by the following probabilities:
   - P_a: This represents the likelihood of attacking the honeypot. P_a affects the transition from state W to state C. When a honeypot is in state W and is ready to execute the attackers' commands (action A), it will be part of a botnet (move to state C) when having an attack attempt, which may occur with the probability of P_a.
   - P_d: This represents the likelihood that botmasters reveal the true nature of the honeypot and remove it from their botnet. Honeypot operators can consider their honeypots as disclosed when receiving a *kill* command or when losing interaction with the botmasters for a long period of time.

C. Transition Rewards

The transitions between states, including self-transitions, are associated with the following rewards/costs:

1) C_O (Operation cost): This cost represents different factors that are needed to deploy, run and control the honeypot.
2) I_V (Information value): One of the main purposes of honeypots is to collect information about attackers. Whenever botmasters interact with a honeypot, the latter collects information about the attackers' techniques, codes and tools, which represents a significant source of knowledge for security professionals and the security community.
3) C_R (Reset cost): This represents the cost associated with resetting the honeypots, e.g., cost associated with resetting virtual machines to their initial clean state.
4) C_L (Liability cost): Honeypot operators might become liable for executing the botmasters' commands if such commands include illicit actions such as DDoS or spamming.

The values of the above mentioned costs are always positive and in practice it is logical to assume that they satisfy the following conditions:
$C_O < C_L$ and $C_R < C_L$; security professionals avoid the legal liability due to its high value compared to other costs.
$C_O < I_V$ and $C_R < I_V$; collecting information is the main objective of honeypots. The cost of operating or resetting the

honeypot is less than the value of the collected information.

In Figure 1, the transitions between states are represented as arrows from the beginning state to the end state. Each transition has a label with the format of (action, reward, probability). For example, (R,$-I_V$,1) means that the system transits with probability 1 when action R is used and the associated cost is $-I_V$. The system starts in state W, i.e., it waits for an attack by botmasters which occurs with a probability P_a. Once it is under attack, the honeypot can choose whether to allow the attacker to compromise it (action A) or not (action N). If the attack is allowed to succeed, the honeypot will be compromised and will join the botnet (state C). Otherwise, it performs a self transition into state W. In state C, the honeypot can hide its true nature by executing the botmaster commands (action A) which makes the botmasters think that the honeypot is a real machine. Consequently, more information (I_V) can be gained. However, this comes with risking the possibility of being legally liable if the honeypot participates in illegal actions (C_L). If the honeypot chooses not to execute the botmaster's commands (action N), there is a probability of disclosure (P_d) when the botmasters send test commands. This may lead the botmasters to disconnect the honeypot form the botnet and cause loss of information (I_V). If the honeypot is disclosed, it moves to state D and stays there until it is reset to the starting state W (action R). Also, the honeypot can be reset to the starting state W from any state to be ready and waiting for a new botmaster's attack. This reset action comes with the cost of C_R. Figure 1 depicts the developed MDP model.

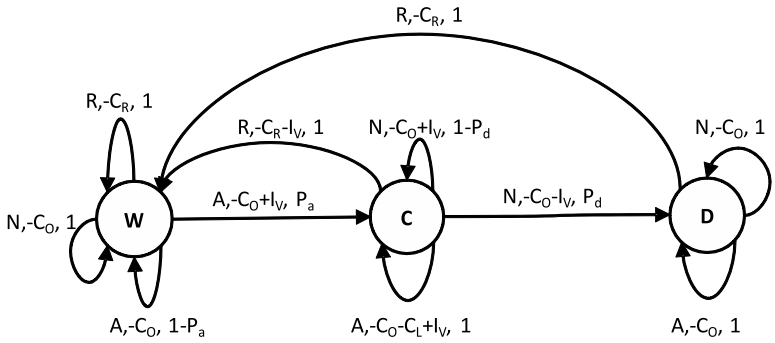


Fig. 1. The developed MDP model for the honeypot interaction with botmasters.

D. Analysis of the Developed Model

A recurrent MDP can be analyzed over either finite or infinite planning horizon. To simplify our analysis, in this work we assume infinite interactions between the honeypot and the botmaster. In this case, different approaches can be used to calculate the gain of MDP model [17]. In our work, we calculate the *expected average reward per period* using the product of steady state probability and the reward vectors as described in [17], considering a discount factor $d = 1$. In what follows, we provide the equations that are used to calculate the gain obtained from choosing a specific policy, i.e., a specific action for each state. Let *N* be the number of states in a MDP and let *P* denote the matrix of transitions probabilities

$$P = \begin{bmatrix} p_{11} & p_{12} & \dots & p_{1N} \\ p_{21} & p_{22} & \dots & p_{2N} \\ .... & & & \\ p_{N1} & p_{N2} & \dots & p_{NN} \end{bmatrix} \tag{1}$$

where p_{ij} denotes the probability of transiting from state *i* to state *j*.
The vector of steady state probabilities is given by

$$\pi = [\pi_{\ 1} \quad \pi_2 \quad \dots \quad \pi_N] \tag{2}$$

where

$$\sum_{i=1}^{N} \pi_i = 1 \tag{3}$$

To calculate π, we need to solve the following set of equations:

$$\pi = \pi \times P \tag{4}$$

with consideration of equation (3). Let *R* denote the matrix of rewards

$$R = \begin{bmatrix} r_{11} & r_{12} & ... & r_{1N} \\ r_{21} & r_{22} & ... & r_{2N} \\ ... & & & \\ r_{N1} & r_{N2} & ... & r_{NN} \end{bmatrix} \tag{5}$$

where r_{ij} represents the reward for transiting from state *i* to state *j*.
The reward vector

$$q = [q_1 \quad q_2 \quad ... \quad q_N]$$

is given by

$$q_i = \sum_{j=1}^{N} p_{ij} r_{ij} \quad , \quad i = 1, 2, ..., N \tag{6}$$

The gain is calculated as

$$G = \sum_{i=1}^{N} \pi_i q_i \tag{7}$$

After calculating *G* for each policy, the optimal policy can be determined by comparing the gains of all policies and selecting the one with the maximum value.

IV. The Optimal Policy

In this section, we discuss all possible policies for the honeypot and determine the optimal one under different assumptions. In our model, we have three states with three possible actions in each state. Thus, the combination of all possible policies results in 27 possible policies. However, as will be explained below, we do not have to investigate all of these policies since some of them are dominated by others.
At state D the only action we should consider is R; a honeypot that has been disclosed by the botmaster is of no use to the honeypot operators and must be reset. Also, when the honeypot is in the waiting state (W) and chooses action N or R it will not be able to join the botnet and cannot provide security professionals with useful information. So the only logical action at state W is to allow the botmasters to compromise the honeypot (action A). Thus, we are left with only three policies to choose from, i.e., policies in which the honeypot always chooses action A at state W and action R at state D. We denote these policies by the name of the action used at state C:

- *Policy A*: Use action A at state C.
- *Policy N*: Use action N at state C.
- *Policy R*: Use action R at state C.

To decide the optimal policy for the honeypot, we calculate the gain obtained in each policy using equation (7) and denote it as G_s where $s \in \{A, N, R\}$. By calculating the gain for each of the three policies, we have

1) *Policy A*:

$$G_A = I_V - C_O - C_L \tag{8}$$

The gain obtained by applying this policy can be positive only if $I_V > C_O + C_L$. This can be the case where the collected information is important, e.g., valuable cyber intelligence information, or in the case of low legal liability, e.g., where liability can by reduced by obtaining a court permit.

2) *Policy N*:

$$G_N = \frac{(P_a - P_a P_d) I_V - (P_a + P_d) C_O - P_a P_d C_R}{P_a + P_d + P_a P_d} \tag{9}$$

The value of G_N becomes smaller when P_d increases. This is because the botmasters are more likely to disclose the honeypots and remove them from their botnets, which reduces information captured by the honeypots.

3) *Policy R*:

$$G_R = \frac{-C_O - P_a C_R}{1 + P_a} \tag{10}$$

G_R always has a negative value that represents a loss for honeypot. Thus the policy R can be excluded since policy N always provides a better reward to the honeypot.

Deciding the optimal policy requires the knowledge of all the system parameters, i.e., costs and probabilities. Although it is possible to estimate all costs such as the cost of running, maintaining and resetting the honeypot, the cost of information (based

on its importance), and the cost of liability (based on the illegal action that botmasters are performing), determining the values of P_a and P_d is a relatively harder task. The probability of attack P_a can be estimated using experimental data collected over a sufficient period of time. However, estimating the probability of disclosure P_d is not easy since it is under the control of the botmasters. On the other hand, analyzing the developed model may provide the security professionals with some guidelines to determine the optimal policy.

For example, suppose the system parameters have the values of ($C_O = 1, C_R = 2, I_V = 20, C_L = 15$ and $P_a = 0.5$). Figure 2 shows the optimal policy for different values of P_d. In this scenario, security professionals must select policy N when the probability of disclosure is assumed to be less than 0.4, and select Policy A when a higher value of P_d is assumed. As mentioned earlier, policy R cannot be optimal. When the probability of disclosure is low, honeypots will be able to stay for a longer time in botnets and collect more information even when choosing the action of not executing the botmasters' commands. However, with higher values of P_d, the optimal policy is to allow botmasters to execute their attacks from within the honeypot since this will hide the honeypot true nature.

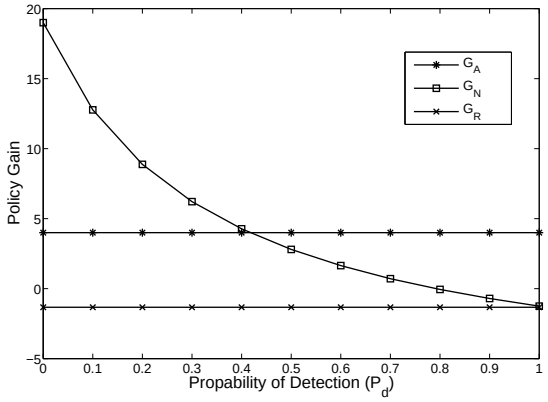


Fig. 2. Changes in the gains (G_A, G_N, and G_R) as a function of probability of disclosure (P_d).

V. Modeling Uncertainty of the Honeypot State

When using MDP, the current state of the honeypot, i.e., W, C, or D, must be known to the honeypot operators in order to decide the best action to use. To determine the system current state, the honeypot needs to look for evidence and/or observations and interpret them accordingly. Some evidence can be deterministic, while others may have different interpretations that lead to uncertainty about the system inner state and, consequently, lead to following non-optimal actions. For example, the absence of suspicious activities while the honeypot has not joined any botnet yet assures the operators that the honeypot is in state W. On the other hand, the absence of botmasters' activities while the honeypot is in state C can be either due to the fact that the botmasters are performing other tasks, e.g., expanding their botnets, or because the honeypot true nature was disclosed by the botmasters. Thus, in this case, it is not possible for the honeypot operators to decide whether the system is in state C or in state D. This uncertainty about the system state cannot be modeled using MDP. In this case, a more general concept, that is able to handle such uncertainty and yet is still capable of determining the optimal policy, is needed. A Partially Observable Markov Decision Process (POMDP) [18], which uses observations to calculate the probability of being in each of the system states, has those capabilities and can be applied to our model.

A. Partially Observable Markov Decision Processes

The definition of POMDP is similar to MDP with the addition of two parameters:

1) A finite set of observations $\mathcal{O}$.
2) An observation function
   O: $A \times S \mapsto [Pr(O_1), Pr(O_2), ...], \sum_i (Pr(O_i) = 1$

In POMDP, after executing action a_{j-1} the system state s_j may not be determined. Instead, we receive an observation O_j and use it to calculate the *probability* of being in each state. The probability distribution over all system states is called the *belief state*. Solving a POMDP involves converting it into MDP problem by replacing the system state with the belief state [18]. However, this solution contradicts with the definition of MDP in two aspects:

1) Maintaining The Belief State:
   The entire history of actions and observations is required to maintain the belief state updated. This violates the Markovian

property which requires that the next state must depend only on the current state and current action. However, as described in [18], the Bayes rule can be used to update the belief state. By knowing the belief value $b(s)_t$ for state s at time t, the taken action a_t and the received observation O_t, the new belief value $b(s)_{t+1}$ can be calculated as:

$$b(s)_{t+1} = \frac{Pr(o|s, b_t, a) \times Pr(s|b_t, a)}{Pr(o|b_t, a)} \tag{11}$$

This enables us to use the belief state as our state set, which converts the POMDP problem into a fully observable MDP.

2) Continuity of Belief State Space:
It is impractical to find the optimal solution for all belief states as their space is continuous and has infinite number of possible states [18]. To overcome this problem, approximation algorithms [19] [20] are used to find an approximation to the optimal solution. All algorithms use value iteration to calculate the approximated solution [18] starting at an initial value function ***V(b)*** for the initial belief state, and then iterating using equation (12) as follows:

$$V'(b) = max_a[\sum_s b[s]R[s,a] + \Upsilon \sum_o Pr(o|b,a)V(b')] \tag{12}$$

where $V'(b)$ is the improved value function for belief state b, Υ is the future discount factor and $V(b')$ is the value of the resulting belief state. The differences between the algorithms used to solve the POMDP reside in the way they sample the belief space to reach the optimal solution [19] [20].

B. POMDP Model

To model uncertainty, we can include different observations to the system and use them in solving the resulting POMDP problem. For example, we can have the following three observations to monitor and calculate the system belief state:

- Unchanged Honeypot: Honeypots are used only to collect information about attackers and have no productivity purpose. When attackers penetrate honeypots, they leave traces, e.g., changes in log files, downloaded files and other activities. If no changes are observed to the honeypot, this indicates that it is still in the waiting state W. We refer to this observation as $Unchanged$.
- Absence of Botmasters' Commands: Botmasters are supposed to make use of their victim machines after turning them into bots. After compromising the honeypot, if botmasters do not send commands to the honeypot, it is possible that they have detected its true nature and disconnect it from their botnet. However, it is also possible that botmasters are busy for a while doing other things such as compromising other machines to expand the botnet. Thus, this observation leads to uncertainty in determining the honeypot state, whether it is in state C or in state D. We refer to this observation as $Absence$.
- Receiving Botmaster Commands: When the honeypot receives commands from the botmasters, after being compromised, it is clear that they still consider it as a part of their botnet. This indicates that honeypot is not disclosed yet and still in state C. We refer to this observation as $Commands$.

VI. Simulation Results

In this section, we present and discuss our simulation results. We show how to determine the honeypot optimal policy based on different configuration parameters, and the effect of these parameters on the honeypot outcome, i.e., expected reward. We explain how to solve our model by using an example of high liability cost scenario, and we show the effect of changing the probability of attack, P_a, and the probability of disclosure, P_d, on both the expected reward and the optimal policy.
To solve the POMDP model, we use the Approximation POMDP Planning Toolkit (APPL) [21] which allows us to find the optimal policy, generate an optimal policy graph, and simulate its expected reward. The APPL toolkit uses Successive Approximations of the Reachable Space under Optimal Policies (SARSOP) algorithm [20] to approximate the solution of the POMDP model. In short, SARSOP is one of the algorithms that focuses on reachable points of the belief space. In general, most points of the continuous belief space cannot be reached starting from a given initial belief. This makes the sampling more efficient and helps solving the POMDP in a shorter time, especially for problems with larger sets of states. Furthermore, SARSOP tries to determine the optimally reachable belief space (the space that contains only the points needed for optimal solution) by applying a technique of learning-enhanced exploration and bounding [20].

A. Scenario of High Liability Cost

In what follows, we assume the following parameters:

1) Transition Probabilities: The probability of attack P_a=0.7 and the probability of disclosure P_d=0.6
2) Costs: The operation cost $C_O = 1$, the reset cost $C_R = 2$, the information cost $I_V = 10$, and the liability cost $C_L = 15$.

3) Observation Probabilities:
- When the system is in state W: The only observation that exists is $Unchanged$ as the honeypot has not joined any botnet yet. In this case, $Pr(Unchanged) = 1.0$, $Pr(Commands) = 0.0$, and $Pr(Absence) = 0.0$.
- When the system is in state C: When compromised, observation $Unchanged$ does not happen. Only the other two observations ($Commands$ and $Absence$) can be observed. In this case, $Pr(Unchanged) = 0.0$. We assume that $Pr(Commands) = 0.7$, and $Pr(Absence) = 0.3$.
- When the system is in state D: Only the observation $Absence$ can be observed as no further commands from the botmaster are received. Also $Unchanged$ cannot be observed as the honeypot has already been compromised. Thus in this case we have $Pr(Unchanged) = 0.0$, $Pr(Commands) = 0.0$, and $Pr(Absence) = 1.0$.

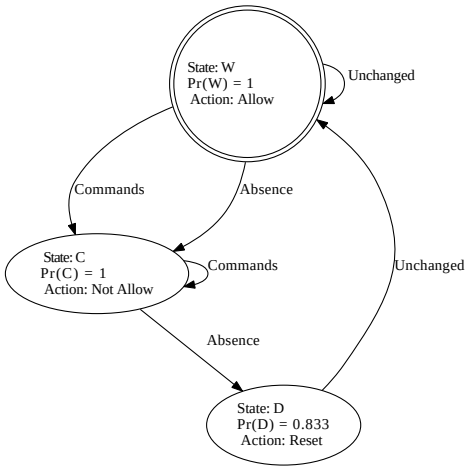


Fig. 3. An example for a honeypot optimal policy graph representation in a high liability scenario ($P_a = 0.7, P_r = 0.6$)

Figure 3 shows the optimal policy for the assumed set of parameters. The system starts in state W with probability 1 and chooses the optimal action A to allow botmasters to compromise the honeypot. If the honeypot is compromised, the system will certainly transit to state C with probability 1, in which the optimal action is N, and stays there as long $Commands$ is observed. Upon receiving observation $Absence$, the system is considered to be in D, with probability of 0.833, in which the optimal action is to reset the honeypot to its initial state W. To summarize, in this scenario, the optimal policy for the honeypot is to use action A when observing $Unchanged$, to use action N when moving from state C (regardless of the observations) and to use action R when having the observation $Absence$ after using the action N.

B. Determining the Optimal Policies and Calculating the Expected Rewards for Different Values of P_a and P_d

In what follows, we study the effect of the parameters change on the expected reward. In particular, we investigate the effect of changing the probability of attack P_a and the probability of disclosure P_d on the expected reward of the system. Figure 4 shows the changes of the expected reward for a particular set of cost values when both P_a and P_d change in the range [0,1]. As depicted in the figure, the value of the expected reward increases when the probability of attack increases. A higher P_a means more attacks are launched to compromise the honeypot and consequently more information is collected by the honeypot at each time interval, i.e., the honeypot will spend less time in W state and be able to gain the reward I_V more often. On the other hand, when the probability of disclosure P_d increases, the expected reward decreases. A higher P_d means that the botmasters are more likely to disclose the true nature of the honeypot and remove it from their botnet, which makes the honeypot less efficient because of the reduction in the collected information. We also notice that higher expected rewards come with a low probability of disclosure even when P_a has a low value. This is due to the fact that with low disclosure probability, honeypots will prolong their stay in the botnet and collect more information in the long term. Security professionals should consider this result when trying to attract more attackers by increasing P_a, e.g., by increasing the attack surface, as this may draw the attackers' attention to consider this machine as a possible honeypot and consequently increase P_d. A balance between P_a and P_d is important when the honeypot is required to stay in the botnet for a longer time.

Figure 5 shows the optimal policies for two sets of parameters that differ only in the value of probability of attack (in Figure 5-a, P_a=0.1 and in Figure 5-b, P_a=0.5). As depicted in Figure 5-a, due to the low value of P_a, the system is trying to capture more information by choosing the action N when the probability of being in D is higher than the probabilities of being in other states. Based on the received observation, the system may be in state C and collect more information or in state D (with higher probability). If the probability of being in D is very high then the system chooses the action R and reset to

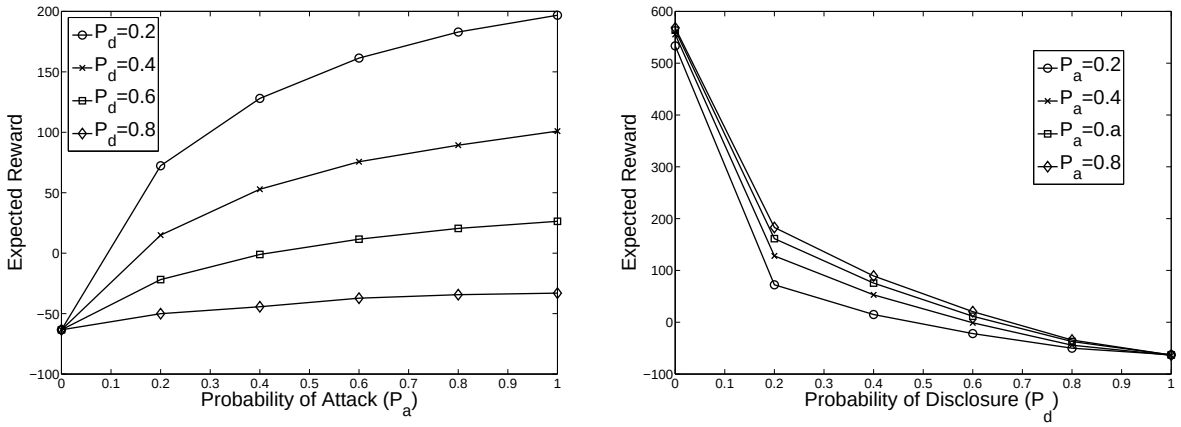
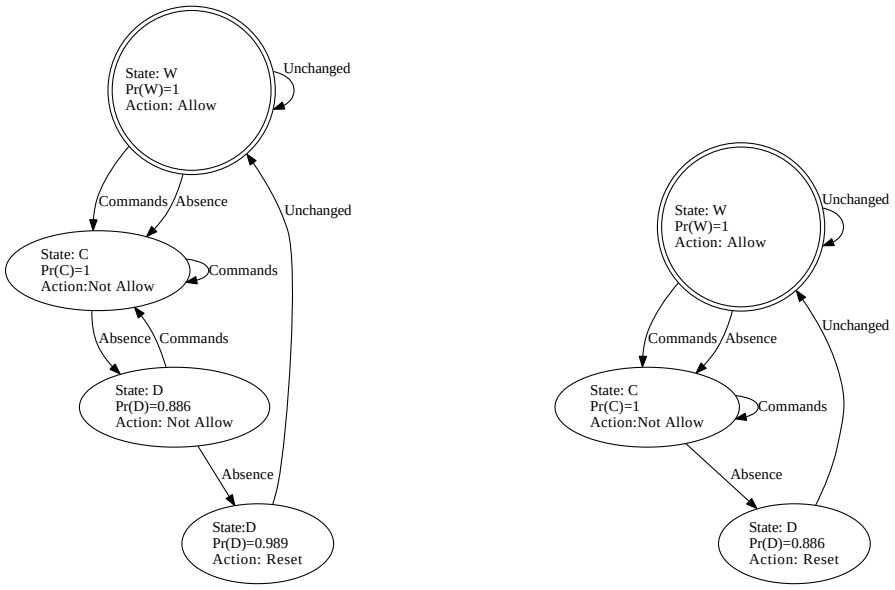


Fig. 4. The system expected reward increases with higher probability of attack, P_a, and decreases with higher probability of disclosure, P_d.



(a) P_a=0.1 (b) P_a=0.5

Fig. 5. Examples for the optimal policy graph for different values of probability of attack, P_a.

the initial state W. For $P_a = 0.5$, in Figure 5-b, the system directly chooses the action R when the probability of being in D is higher than the probability of being in other states. This is due the higher probability of attack, which makes it more rewarding to reset the system and wait for new attacks rather than hoping for the current attackers to make new interactions with the honeypot after observing the absence of their commands.

C. Determining The System Expected Reward for Different Observations' Probabilities

In what follows, we study the impact of changing the probabilities of observations on the expected reward of the system. In particular, we study the effect of changing the probability of having observations: $Commands$ and $Absence$, after executing the action N in state C. To do so, we use the values of the remaining parameters as follows where: $C_O = 1$, $C_R = 2$, $C_L = 15$, $I_V = 10$, $P_a = 0.8$, $P_d = 0.5$. In this scenario, it is expected to have either observation $Commands$ or observation $Absence$ in state C with probabilities $Pr(Commands)$ and $Pr(Absence)$ consecutively, where $Pr(Commands) + Pr(Absence) = 1$. Figure 6 shows the effect of changing these probabilities on the expected reward of the system.

From the figure, we can notice that the expected reward of the system increases with higher probability of having observation $Commands$. This is due to the fact that higher probability for observing $Commands$ represents more possible interactions with the botmasters. Thus, our model is expecting the system to receive I_V with more probability at each interaction.

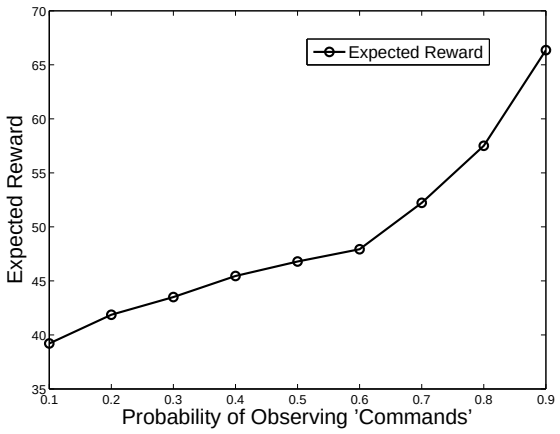


Fig. 6. The system expected reward increases with higher probability of observing $Commands$ in state C after executing the action N

VII. Conclusions

Introducing honeypots has been a milestone for the security community. Honeypots allow security professionals to track, capture, and learn attackers' tools and techniques. However, attackers have also started to fight back and develop tools and techniques to disclose honeypots. In this work, we used Markov decision processes and partially observable Markov decision processes to model the interaction between botmasters and honeypots. The analysis of our model confirms that exploiting the honeypots legal liability allows botmasters to have the upper hand in their conflict with honeypots because the honeypot operators cannot estimate the state of the honeypot, as determined by the botmaster, with full certainty. Despite this deficiency in current honeypot designs, the developed models can be used to help security professionals determine the optimal response to botmasters commands and prolong the honeypot lifetime inside the botnets.

References

[1] N. Provos. A virtual honeypot framework. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, pages 1–14. USENIX Association, 2004.
[2] P. Ferrie. Attacks on more virtual machine emulators. Symantec Advanced Threat Research, 2006.
[3] N. Krawetz. Anti-honeypot technology. *IEEE Security & Privacy Magazine*, pages 76–79, 2004.
[4] C. Zou and R. Cunningham. Honeypot-aware advanced botnet construction and maintenance. In *Dependable Systems and Networks*, pages 199–208, 2006.
[5] M.L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc., 1994.
[6] X. Fu, W. Yu, D. Cheng, X. Tan, K. Streff, and S. Graham. On recognizing virtual honeypots and countermeasures. In *2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, pages 211–218, 2006.
[7] O. Hayatle, A. Youssef, and H. Otrok. Dempster-shafer evidence combining for (anti)-honeypot technologies. *Information Security Journal: A Global Perspective*, pages 306–316, 2012.
[8] G. Shirazi, P Kong, and C. Tham. Cooperative retransmissions using Markov decision process with reinforcement learning. In *IEEE 20th Interantional symposium on Personal, Indoor and Mobile Radio Communications*, pages 652–656, 2009.
[9] Z. Abbas and F. Li. Energy optimization in cellular networks with micro-/pico-cells using Markov decision process. *European Wireless, 2012. EW. 18th European Wireless Conference*, pages 1–7, 2012.
[10] S. Jha, O. Sheyner, and J. Wing. Two formal analysis of attack graphs. In *Computer Security Foundations Wrokshop*, pages 49–63, 2002.
[11] M. Taibah, E. Al-Shaer, and R. Boutaba. An architecture for an email worm prevention system. In *Securecomm and Workshops, 2006*, pages 1–9, 2006.
[12] O. Kreidl. Analysis of a Markov decision process model for intrusion tolerance. In *Dependable Systems and Networks Workshops*, pages 156–161, 2010.
[13] J. Liu, F.R. Yu, C. Lung, and H. Tang. Optimal combined intrusion detection and biometric-based continuous authentication in high security mobile ad hoc networks. *IEEE Transactions on Wireless Communications*, pages 806–815, 2009.
[14] M. Petrik, G. Taylor, R. Parr, and S. Zilberstein. Feature selection using regularization in approximate linear programs for Markov decision processes. In *The 27th International Conference on Machine Learning*, pages 871–878, 2010.
[15] M. Feily, A. Shahrestani, and S. Ramadass. A survey of botnet and botnet detection. In *Emerging Security Information Systems and Technologies, 2009. SECURWARE '09*, pages 268 –273, 2009.
[16] G. Wagener, R. State, A. Dulaunoy, and T. Engel. Self adaptive high interaction honeypots driven by game theory. In *Symposium on Stabilization, Safety, and Security*, pages 741–755, 2009.
[17] T. Sheskin. *Markov Chains and Decision Processes for Engineers and Managers*. CRC Press, 2011.
[18] A. Cassandra, L. Kaelbling, and M. Littman. Acting optimally in partially observable stochastic domains. In *American Association for Artificial Intelligence*, pages 1023–1028, 1994.
[19] A. Cassandra. Pomdps: Who needs them? www.pomdp.org/pomdp/talks, 2003. Online; accessed 27-November-2012.
[20] H. Kurniawati, D. Hsu, and S. Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems IV*, 2008.
[21] Y. Du, D. Hsu, X. Huang, H. Kurniawati, W. Sun Lee, S. Ong, and S. Png. Approximate pomdp planning software (appl). http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl. Online; accessed 19-June-2012.