ANALYTICAL MODELS FOR THE INTERACTION BETWEEN BOTMASTERS AND HONEYPOTS

OSAMA HAYATLE

A THESIS

IN

THE CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE IN INFORMATION SYSTEMS

SECURITY

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

MARCH 2013

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By:	**Osama Hayatle**
Entitled:	**Analytical Models for the Interaction Between Botmasters and Honeypots**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Information Systems Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Ben Hamza ______________________	Chair
Dr. Khaled Galal ______________________	Examiner
Dr. Chadi Assi ______________________	Examiner
Dr. Amr Youssef ______________________	Supervisor
Dr. Hadi Otrok ______________________	Supervisor

Approved by ______________________________

Chair of Department or Graduate Program Director

Date February 19, 2013

Dr. Robin Drew, Dean

Faculty of Engineering and Computer Science

ABSTRACT

Analytical Models for the Interaction Between Botmasters and Honeypots

Osama Hayatle

Honeypots are traps designed to resemble easy-to-compromise computer systems in order to tempt attackers to invade them. When attackers target a honeypot, all their actions, tools and techniques are recorded and analyzed in order to help security professionals in their conflict against the attackers and the botmasters. However, botmasters might be able to detect honeypots. In particular, they can command compromised machines to perform illicit actions in which the targeted victims work as sensors that measure the machine's willingness to perform these actions. If honeypots were designed to completely ignore these commands, then they can be easily detected by botmasters. On the other hand, full participation by honeypots in such activities has its associated costs and may lead to legal liabilities. This raises the need for finding the optimal response strategy needed by honeypots in order to prolong their stay within botnets without exposing them to liability.

In this work, we show that current honeypot architectures and operation limitations may allow botmasters to uncover honeypots in their botnet. In particular, we show how botmasters can systematically collect, combine and analyze evidence about the true nature of the machines they compromise using Dempster-Shafer theory. To determine the currently available optimal response for honeypots, we provide a Bayesian game theoretic framework that models the interaction between honeypots and botmasters as a non-zero-sum noncooperative game with uncertainty. However, the solution of the game shows that botmasters always have the upper hand in the conflict with honeypots since botmasters can update their belief about the true nature of the opponents and consequently act optimally based on the new belief value. This motivated us to investigate a better strategy that enables honeypots to maximize their outcome by optimally responding to the probes of the botmasters. In particular, we provide a Markov Decision Processes model that helps security professionals to determine the optimal strategy that enables the honeypots to prolong their stay in the botnets while minimizing the cost of possible legal liability.

Throughout this thesis, we also provide different scenarios that illustrate and support our proposed analysis and solutions.

Acknowledgements

I would like to thank each and every person who made this work possible and gave me the chance to accomplish a dream.

First of all, I would like to thank my supervisors, Dr. Amr Youssef and Dr. Hadi Otrok, for their guidance, support, time, and the effort they spent in making me proceed into this work step by step. I was lucky for having such caring and knowledgeable supervisors who made this work possible and meaningful.

My warmest thanks to my parents, the role models in my life who gave me everything I have.

My very special thanks to my wife Sana for being with me while taking care of our family all the time.

Finally, I would like to thank my colleagues at CIISE for the great time we spent together and for their help and support, especially Abdel Alim and Aleksandar.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we present a brief outline of our research work. We start with an overview section introducing our topic, and then, we present our motivation for selecting this topic. After that, we present our objectives and contributions. Finally we conclude this chapter by presenting the organization of the remainder of this thesis.

1.1 Overview

Honeypots [32] have become an essential tool for security professionals because of their magnificent contributions in disclosing the underworld of cybercrimes. Honeypots are non-productive systems used to lure hackers and botmasters by allowing them to gain access to seemingly compromised systems. When botmasters interact with honeypots, all their actions, tools, and techniques are captured, which enables security professionals to have a better understanding of the tools and techniques developed and used by hackers and botmasters. The ease of use and the valuable information they provide helped promoting the use of honeypots for different purposes including security research and attacks' mitigating. However, this valuable source of information is being threaten by new techniques that focus on disclosing and exploiting the specific characteristics of honeypots [22]. For example, honeypots are usually deployed as virtual machines to facilitate controlling them, this leads attackers to probe virtual environments [19] looking for possible honeypots. Detecting honeypots and anti-honeypots techniques became an important area of research to uncover the weak sides of honeypots, which enables improving them and build better honeypots that can avoid attackers' detection techniques.

1.2 Motivation and Objectives

If hackers are able to detect honeypots, then attacking honeypots may also provide hackers and botmasters with valuable information about their observers the same way that honeypots were designed to provide security professionals with valuable information about botmasters [32]. Thus, hackers are actively working on finding ways to detect honeypot traps. In fact, some anti-honeypot methods have already appeared on the web (e.g., see www.send-safe.com). It should also be noted that it is not only hackers that are investigating potential weaknesses of honeynets. Some security professionals have recently looked at different weak sides of honeypots and investigated countermeasures against them ,e.g., [6, 13]. These studies are aimed to warn the security community of potential pitfalls of current honeypot technologies, draw the attention of honeypot developers to possible limitations and deficiencies and help develop new honeypot systems with anti-detection techniques. Zou and Cunningham [51] suggested a software and hardware independent methodology to disclose honeypots and remove them from botnets. In this methodology, the botmaster commands the bot in the compromised machine to execute some illicit actions ,e.g., spamming or making continuous web requests that look like a DDoS attack. The target of these actions is a machine (or more) owned by the botmaster which serves as a sensor to detect the execution of the attack. The work in [51] assumes that honeypots do not respond to these kinds of commands by the botmasters. However, if honeypots are designed to completely ignore these commands, then they can be easily detected by the botmasters. On the other hand, full participation by the honeypots in such activities has its associated cost and may lead to legal liabilities [32]. This raises the need for considering the following question: *What is the best response strategy that the honeypot can follow in order to evade detection by botmasters without completely sacrificing the liability?* Furthermore, the execution of such tests by botmasters is bounded to different types of constraints such as firewall settings that might prevent normal bots from always cooperating [46]. Such constraints can lead botmasters to misjudge normal bots as honeypots and drop them from the botnet. To avoid such a false negative decision, botmasters need to run their tests several times in order to build a belief evaluation of the machine nature and then, only when the evaluation belief reaches a specific threshold, botmasters should remove the machine from the botnet. Thus, the question that raises in this context is: *How many test commands are required to differentiate between a normal machine and a honeypot with enough level of* *confidence?*

The objective of this work is to provide mathematical models that help analyzing and understanding the interaction between the botmasters and the honeyptos. Utilizing these models allows the operators of honeypots to optimize their response in order to avoid the botmasters' anti-honeypot detection techniques and consequently prolong the lifetime of the honeypots inside the botnets while minimizing the risk of being legally liable.

1.3 Proposed Approach

In this thesis, we first investigate the botmasters' ability to disclose honeypots in their botnets. In particular, we present a mathematical model, based on Dempster-Shafer theory of evidence, for detecting honeypots. In this model, botmasters collect multiple evidence about the machine they are interacting with during different phases of the compromising process. Each evidence is assigned a basic belief assignment value (BBA) that reflects its support for each machine type, i.e., honeypot or normal. Then, by using Dempster rule of combination, botmasters combine all evidence in order to calculate the final belief of the machine's true nature. The analysis of our DS model shows that botmasters can determine the true nature of compromised machines with relatively high certainty. Furthermore, this approach can be used by botmasters to overcome the technique suggested in [41] where the authors argued that equipping real machines with fake evidence that make them appear as honeypots can deceive attackers and make them avoid such machines. The obtained results motivated us to formally analyze the possible interaction between botmastes and honeypots and to explore the current available response strategy for honeypots. For this purpose, we modeled the botmasters-hoenypots interaction as a non-cooperative non-zero-sum theoretical game. In this game, both the botmaster and the honeypot are seeking to maximize their outcome by choosing the best available strategy. More precisely, by solving this game model, we are able to determine the optimal percentage by which honeypots must contribute in executing the commands of botmasters in order to prolong their lifetime in the botnet. This also allows us to determine the optimal percentage of test commands, i.e., the percentage of fake attack commands used by botmasters to disclose the true nature of their compromised machines. However, the analysis of our game model shows that botmasters can use a belief function to update their belief about the true nature of their opponent, i.e., whether it is a honeypot or a normal machine, and consequently, modify their optimal percentage of tests, which leads them to disclose honeypots and remove them from the botnet. On the other hand, honeypots cannot act optimally as they are not able to distinguish actual attacks from fake ones. These results motivated us to investigate a better response strategy for honeypots. To achieve this objective, we developed a Markov decision process model for the interaction between honeypots and botmasters. In this model, honeypots maximize their outcome by selecting the optimal action at each state of the model. The set of optimal actions represents the honeypot optimal strategy for responding to the probes of botmasters. This optimal strategy enables honeypots to have a longer stay inside botnets while reducing the cost of the legal liability.

1.4 Contributions

In this thesis, we have achieved a set of contributions that can be summarized as follows:

- We showed that attackers can use a systematic methodology (Dempster-Shafer theory of evidence) to collect pieces of evidence from honeypots during different phases of compromising and combine these evidence in order to determine the true nature of the machines they are interacting with. This methodology can also be used to determine the true nature of normal machines equipped with fake evidence for the purpose of making them appear as honeypots.

- We developed a game theoretic model that can be used by security professionals to determine the best response to attackers probes by calculating the honeypot optimal contribution percentage (the percentage of executing the commands of the botmasters). We also investigated botmasters' best strategy to detect honeypots and showed that botmasters can update their belief about the true nature of their opponent such that they can disclose the honeypots in their botnte.

- We modeled the interaction between the botmasters and the honeypot by a Markov Decision Process (MDP). This decision-making model enables the operators of the honeypots to maximize their gain when interacting with botmasters by selecting the optimal strategy to respond to the probes of botmasters, which enables the honeypots to have a longer stay inside the botnet while avoiding the legal liability of participating in illicit actions.

1.5 Thesis Organization

The remaining of the thesis is organized as follows. In chapter 2 we introduce some preliminaries related to our work which include background about game theory, Dempster-Shafer theory, and Markov Decision Processes. In chapter 3, we show how attackers can collect different pieces evidence during the compromising process, and then use Dempster-Shafer theory to combine these pieces of evidence to calculate the belief value about the true nature of the investigated machine. In chapter 4, we model the interaction between honeypots and botmasters using a game theoretical framework, in which both parties (players) of the game seek to maximize their outcomes by selecting the best strategy considering the possible choices of each other. In chapter 5, we use Markov Decision Processes to represent honeypots interaction with botmasters. In this MDP model, honeypots are able to determine their optimal policies that allow them to maximize their outcome by prolonging their stay in botnets while avoiding legal liabilities. Finally, we discuss our findings and present our conclusions and future work in chapter 6.

Chapter 2

Preliminaries

In this chapter, we present an overview about botnets, honeypots and the mathematical tools we use to model the interaction between botmasters and honeypots.

2.1 Botnets Overview

A botnet is an army of compromised computers [29] controlled by a bot herder, also known as the botmaster, and used to perform illegal Internet-based malicious activities such as spam spreading, identity theft, and distributed denial of service (DDoS) attacks. Botnets have introduced a new kind of organized underground business that allows criminals to rent a botnet, or part of it, [2] to be used for illicit activities. This potential financial revenue has raised botnets to become one of the most important threats in the Internet security world [51]. The number of bots, i.e., compromised machines, of the botnet ranges from few hundreds to millions [50]. Each bot receives the commands of the botmaster through a

Command and Control (C&C) infrastructure [40]. Depending on C&C architecture, botnets are classified into two main categories [40]:

- Centralized (Hierarchical) Botnets: In this architecture, botmasters communicate with the members of their botnet through multiple bot-controllers. As shown in Figure 1 , the botmaster sends the commands and codes to the bot controllers which, in turn, distribute them to the bots. This approach facilitates the botmasters controlling of botnets since they need to communicate with few machines only and it also expedites the communications with the bots as it involves two levels of communications only (botmaster $\rightarrow$ bot-controller $\rightarrow$ bot). This architecture also helps in hiding the identity of the botmasters from the bots, thus, if one of these bots is a honeypot it cannot determine the identity of the botmasters. However this centralized communication architecture has the risk of a single point of failure. If one of the bot contrlloers is taken over, the entire botnet will be lost and the actual identity of the botmaster may be disclosed [40].

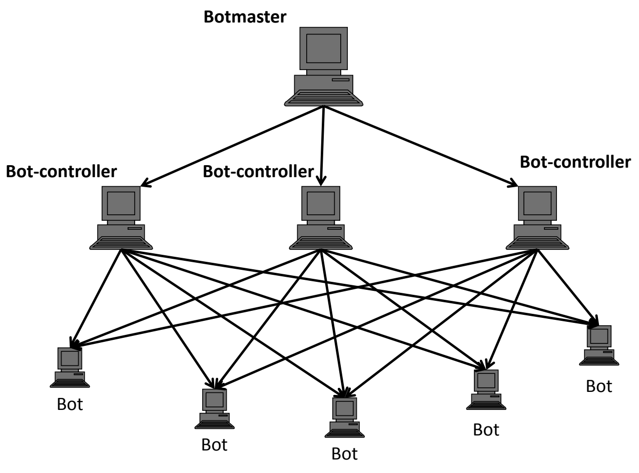


Figure 1: A botnet with a centralized C&C architecture

- Peer-to-Peer Botnets: In this architecture, the botmaster communicates with one of the bots, which spreads the commands to some other bots, which, in turn, spread the commmands to more other bots. There is no specific bots that control others; any bot can spread commands to bots ‘near’ it. This approach hardens the detection of the true identity of the botmasters and avoids the single point of failure. However, such botnets are harder to manage and have slower communications [40]. Figure 2 shows an example of a botnet with peer-to-peer architecture.

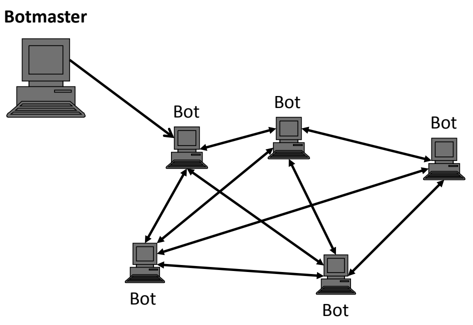


Figure 2: A botnet with a Peer-to-Peer C&C architecture

Honeypots are considered among the most important tools used by security professionals against botmasters. The next section provides a brief introduction to honeypots.

2.2 Honeypots Overview

Honeypots provide security defence against attackers by different means [32]. Their role varies from simply distracting attackers to adaptive honeypots that are able to deceive attackers to reveal more of their tools and techniques [49]. For this, it is hard to precisely define honeypots. According to [45] "A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource". Honeypots classification varies depending on different factors; they can be classified based on complexity, their role or other factors [32]. Based on the complexity, honeypots are categorized into two categories:

1. Low interaction honeypots: They are usually emulators of operating systems and/or services [32]. This emulation limits what honeypots can capture. For example, emulating SSH service allows honeypots to capture login attempts and other SSH services only. For their simplicity, low interaction honeypots are easier to install and maintain. They are usually added as installed software to the host operating systems. They are also safer as attackers do not have access to a full operating system, throughout the honeypot, which limits what attackers can do. On the other hand, the disadvantages of low interaction honeypots are the limitations to capture only known malicious activities that take place on the emulated services and being easier to detect by attackers [33] [21].

2. High interaction honeypots: They are deployed by installing actual operating systems and applications either as virtual machines, which is more common, or as real physical machines [32]. This gives attackers the flexibility to interact with the honeyptos using different means, which enables honeyhpots to capture more information about the attackers' behaviour, tools and techniques. This also enables high interaction honeypots to capture new unknown attacks and provide security professionals with valuable information. On the other hand, high interactions honeypots are more

complex to deploy and maintain which requires more efforts and resources. They also carry another risk; it is possible for attackers to launch malicious activities from within high interaction honeypots. Thus, outgoing traffic must have limitations in order to avoid allowing honeypts to participate in illicit actions [51].

Honeypots can also be categorized based on their role as follows:

1. Production honeypots: These honeypots are used to protect other systems of an organization. Their role is to detect malicious activities and report them, delay or stop attacks, and help in responding to attacks by providing information about attackers and their behaviour [32].

2. Research honeypots: This class of honeypots is used to collect information by security researchers. The collected information can be used to predict new trends in attacks, possible targets and even new vulnerabilities, i.e., zero day attacks [32].

The importance of honeypots and their wide spread explain the new developments in anti honeypot technologies and the importance of the researches conducted to improve this valuable tool. The following section reviews some of the literature in this area.

2.3 Detecting Honeypots

Detecting honeypots has drawn the attention of both attackers and researchers. Attackers aim to detect honeypots in order to avoid them and stay uncovered, while researchers are interested in discovering the weaknesses of honeypots in order to improve them and to better hide these stealthy information resources. Techniques for detecting honeypots focus on exploiting well-known and recognizable honeypot features. Bethencourt *et al.* [6] proposed an attack technique to discover the location of Internet sensors, including honeypots, by probing their publicly published reports. In particular, the authors in [6] suggested to probe a specific range of IP addresses and then check the public reports of Internet sensors. If the IP's of their computers were found then that range must have at least one sensor in it. Then, they divide the IP range into smaller ranges and repeat the test until they locate the targeted sensors. Krawetz [26] introduced a commercial software, *Send-Safe*, that allows spammers to avoid honeypots by using the *Honeypot Hunter* tool which installs a mail server on the honeypot and connects back to itself. If the connection is established and the tool cannot send itself the request, then this machine, which prevents outbound connections, is considered as a honeypot. Ferrie [17] described attackers techniques 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. Zou and Cunningham [51] suggested a software and hardware independent methodology to disclose honeypots and remove them from botnets. In this methodology, the botmaster commands the bot in the compromised machine to execute some illicit actions, e.g., spamming, or making continuous web requests that look like a DDoS attack. The target of these actions is a machine, or more, owned by the botmaster which serves as a sensor to detect the correctness of the attack. Motivated by the fact that attackers are becoming more aware of honeypots and that they generally want to avoid being trapped into them, Rowe *et al* [41] suggested equipping normal machines by misleading evidence to make attackers consider them as honeypots. A rational attacker who believes to be interacting with a honeypot would usually withdraw and leave the machine in order to avoid being captured.

The previous works, among others, motivated us to investigate the interaction between honeypots and attackers using different mathematical tools. In particular, we use Dempster-Shafer theory of evidence, game theory, and Markov decision processes to analyze the interaction between honeypots and botmasters and to gain a better understanding about the abilities of each party. The following sections review these tools in brief.

2.4 Dempster-Shafer Theory

In mathematics, Dempster-Shafer theory of evidence (DS) [42] is a framework used to calculate the belief value, in the range [0 1], for a given hypothesis. In DS theory, we take multiple evidence from different sources to verify the correctness of a given hypothesis. Each evidence supports the questioned hypothesis with a basic belief assignment (BBA), with a value in the range [0,1], which reflects the degree of confidence the evidence provides for the hypothesis i.e., how much the evidence supports the correctness of the hypothesis. After collecting the evidence, Dempster rule of combination is used to calculate the total belief of the questioned hypothesis by combining the BBA values of all evidence into one final belief. In other words, DS theory enables us to determine the probability that a given hypothesis is correct by combining multiple pieces of evidence. The importance of DS theory resides in its ability to handle ignorance. Unlike the Bayesian theory [5], DS theory enables dealing with uncertainty (ignorance), it fits in situations where the prior probability is unavailable. Despite being relatively new, DS theory found its way in many applications including computer security, intrusion detection, and criminal forensics. Otrok *et al.* [35] used DS theory to identify possible attackers in Mobile ad hoc Networks (MANET) by calculating the belief value about the true nature of suspicious nodes, i.e., normal users or attackers. The authors in [35] considered the actions of suspected users as evidence that support each hypotheses, i.e., normal or attacker, with BBA values. By combining different evidence, the authors were able to identify malicious nodes. Chen and Venkataramanan [11] used Dempster-Shafer (DS) theory for intrusion detection in ad-hoc networks. Their methodology is based on using independent nodes to collect different pieces of evidence about the suspected node. To avoid having misleading evidence by dishonest nodes, they built a reputation system in which the belief related to the evidence is calculated based on the node reputation. This way, even when a dishonest node reports a wrong evidence about some other node, the weight of this evidence will have less impact on the final decision. After assigning the belief value for the evidence, the authors used Dempster rule of combination to calculate the final belief about the suspected node. They also compared their work to other voting techniques, such as majority voting and average value, and found that combining evidence by using Dempster-Shafer is more efficient in the existence of dishonest nodes that are trying to change the voting results. In chapter 3 (which is partially published in [21]), we show that botmasters are capable of using DS theory in order to disclose honeypots in their botnets. Botmasters can collect different types of evidence

Table 1: An example for the pay-off table of the *battle of the sexes* game

		Player B	
		Opera	Football
Player A	Opera	3,2	0,0
	Football	0,0	2,3

throughout the compromising process, give each evidence a BBA value that supports one (or both) of machine types, i.e., normal or honeypot, and then combine these evidence using Dempster rule of combinaiton to obtain the belief degree about the true nature of this machine.

2.5 Game Theory

Game theory is a mathematical tool used to analyze decision-making problems between rational players [7]. In such games, players, strategically and rationally, aim to maximize their outcomes (pay-off) by selecting the best strategy from a set of available strategies (called moves or actions). Usually a pay-off table is used to present the pay-off for each player based on the selected strategies. Table 1 shows an example of the pay-off table for a famous theoretical game problem called *battle of the sexes*[1] with two players; wife and husband, where 'Player A' represents the wife.

The classification of theoretical games depends on different factors [34]:

[1]Battle of the sexes is a cooperative non-zero-sum theoretical game. In this game, a couple has agreed to meet but cannot recall if they are attending the opera or a football match, and they realize that none of them can remember where to go. However, it is known for both of them that the husband prefers to go to the football game while the wife prefers to go to the opera. Both would prefer to go to the same place together rather than different ones. The question is, if they cannot communicate, where should they go?

- Players Cooperation: In cooperative games, players help each other in maximizing the total pay-off for all players. On the other hand, in non-cooperative games, players seek maximizing their individual pay-off regardless of the other player's pay-off.

- Sum of the pay-off: In zero-sum games, the pay-off of all players sums to zero, which means players are competing to increase their individual pay-off while decreasing the pay-ff of the other player. Usually, this is the case of non-cooperative games. In non-zero-sum games, the outcomes for players are not related. It is possible to change the outcome for one player without affecting the pay-off of the other player.

- Players set of strategies: In a symmetric games, the pay-off is related to the chosen strategy where both players have the same set of strategies. Thus, the pay-off table for such games is symmetric and players can be swapped without affecting the game. On the other hand, in asymmetric games, players may have different set of strategies and, consequently, have different pay-off values.

- Simultaneous and sequential movements: In simultaneous games, players decide their strategies at the same time, or without prior knowledge for the other player choice. However, sequential games involve turn-by-turn movements, where one player makes a decision before the other. It is still possible, in sequential games, a player does not have a full knowledge about choices of the other player.

- Knowledge of players' actions: In perfect information games, each player is fully aware of the history of the other player movements. In imperfect information games, players may have partial knowledge about their opponent movements, or may even know nothing about it, i.e., zero knowledge.

A game is usually described by more than one of the above factors. For example, the game we use in this thesis is a non-cooperative non-zero-sum game with imperfect information. Other classification criteria may also be used [34]. Game theory has many applications in economics, politics, computer science and other important fields. It can be used at any situation where strategic thinking and planning is critical. Wagener *et al.* [49] developed a self-adaptive honeypot that can tackle attackers to reveal more of their tools. They employed game theory to decide the series of actions that the honeypot allows the attackers to perform and also to decide when to prevent attackers from completing their actions. This enforces hackers to try different methods and, consequently, reveal more of their tools and techniques. Píbil *et al.* [37] developed two game theoretic models that show how to strategically configure the honeypots in organization networks. The first model suggested by [37] is a deception game that aims to attract the attackers to the honeypots rather than any other real server, while the second model adds probing capabilities to the attackers. Both models were solved using Nash equilibrium [34] to determine the strategical use of the honeypts.

In chapter 4 (which is partially published in [20]), we use a game theoretical model to analyze the interaction between honeypots and the botmasters. Both players aim to maximize their pay-off by choosing the best strategy; the honeypots aim to hide their identity while the botmasters aim to distinguish honeypots from real machines in the botnet.

2.6 Markov Decision Processes

Markov Decision Process (MDP) [39] is a mathematical framework used to model automated decision-making where systems are required to automatically decide the best response in different scenarios, such as mobile robots and navigation systems. In brief, a MDP model consists of a set of states that represent different modes of the system, a set of transitions that transit the system from one state to another, and a set of rewards that the system gains (or loses) when transiting from one state to another, including self-state transition. Usually, transitions and rewards are associated with probabilities that affect the transition between states and known as transition probabilities. In MDP models, it is required to have a full knowledge of the system current state. However, in real world situations, it is possible to only have a partial knowledge about the system actual state which leads to uncertainty. To overcome this problem, Partially Observable Markov Decision Processes (POMDPs) are introduced. In a POMDP, we do not consider the system actual state, instead, we consider some evidence or observations that reflect the likelihood of being in each state of the system. In other words, we look for observations that enable calculating the probability distribution of the system states (the probability of being in each one of the states), which is known as the belief state. When using a MDP or a POMDP, we aim to get the maximum reward of the system by choosing the best action for each state (or each belief state in POMDP) such that the total reward of the system is maximized. The set of the optimal actions is called as the *Optimal Policy*.

Markov Decision Processes are widely used as optimization tools for determining optimal strategies in automated systems, e.g., see [44], [1].

Jha *et al.* [24] explained how to interpret attack graphs as MDPs models and solved these models to determine the optimal defence strategy that minimizes the probability of attack success. Taibah *et al.* [47] presented a dynamic defence architecture against email worms where they classified emails into categories based on their threat level, and then used a MDPs 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 [27] employs MDPs to determine the optimal action against attack attempts where it was 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 MDPs and POMDPs, respectively. Liu *et al.* [30] used POMDPs to design a framework for combining intrusion detection and continuous authentication in mobile ad hoc networks (MANETs).

In chapter 5, we use both MDP and POMDP to model the interaction between honeypots and botmasters. The goal of our model is to enable the honeypot to choose the optimal action based on the system state (or system belief state in POMDP) such that it achieves a maximum reward for the entire system. The set of chosen actions represents the optimal policy for the honeypot. This optimal policy enables a longer stay inside the botnet while minimizing the risk of participating in illegal actions.

Chapter 3

Dempster-Shafer Evidence Combining for (Anti)-Honeypot Technologies

In this chapter, we address the botmasters' ability to determine the true nature of a compromised machine. In particular, we show that botmasters can detect hontypots that are trying to hide their identity by mimicking the specifications of normal machines and can detect normal machines that are trying to frightening botmasters using faked evidence that make them appear as honeypots.

3.1 Introduction

Motivated by the fact that attackers are becoming more aware of honeypots and that they generally want to avoid being trapped into them, Rowe *et al.* [41] suggested equipping normal machines by misleading evidence to make attackers consider them as honeypots. A rational attacker who believes to be interacting with a honeypot would usually withdraw and leave the machine in order to avoid being captured. It should be noted, however, that the work in [41] does not consider the possibility that attackers can also collect and analyze evidence that support the fact that they are interacting with a normal machine. In this chapter, we address the possibility of systematically determining the true nature of a compromised machine by botmasters. We show that the current architectures and operation limitations of honeypots allow attackers to systematically collect, combine and analyze evidence about the true nature of the machines they compromise. In particular, we show how systematic techniques for evidence combining such as Dempster-Shafer theory [42] can allow botmasters to determine the true nature of compromised machines with a relatively high certainty. We demonstrate honeypot detection methodology, in which botmasters can detect honeypots by collecting different forms of evidence from different sources, and then systematically combine them using the Dempster-Shafer theory of evidence [42]. The main objective of this chapter is to alert the security community to the following inherent weaknesses in current honeypot design and network defence approaches:

- Attackers can use systematic methodologies to collect pieces of evidence from honeypots and combine them in order to calculate a belief value for the true nature of the machines they are interacting with. This enables botmasters to avoid honeypots and remove them from their botnets.

- Using fake honeypots to protect real machines may not work in practice. Attackers can overcome the idea of misleading them, i.e., planting false information into real machines, by systematically collecting evidence that support the fact that they are interacting with real machines.

3.2 Honeypot Detection Using DS Theory

In this section, we describe the Dempster-Shafer theory (DS) [42] in brief, and then we introduce our DS-based model for detecting honeypots.

3.2.1 Dempster-Shafer Theory of Evidence

In mathematics, the Dempster-Shafer (DS) theory [42] is used to calculate the belief degree, within the range [0,1], for a given hypothesis by combining evidence from different sources. Each evidence must have a belief mass assignment that reflects the support for each possible proposition of the system.

Let $\mathbb{S}$ be the finite set of system propositions, and its power set is $\mathbb{P} = 2^{\mathbb{S}}$. We define a basic belief assignment (BBA) $m : \mathbb{P} \mapsto [0, 1]$ such that $m(\phi) = 0$ and $\sum_{A \in \mathbb{P}} m(A) = 1$.

Each hypothesis $A \in \mathbb{P}$ has two bounds; a lower bound which is called belief (***Bel***) and an upper bound which is called plausibility (***Pl***)

$$Bel(A) = \sum_{B|B \subseteq A} m(B) \tag{1}$$

$$Pl(A) = \sum_{B|B \cap A = \phi} m(B), \tag{2}$$

where the relation between equations (1) and (2) is

$$Pl(A) = 1 - Bel(\overline{A}), \tag{3}$$

where $\overline{A}$ represents the subsets that do not intersect with *A*.

If we have two independent pieces of evidence represented by their BBA values m_1 and m_2 from different sources, then we can combine them using Dempster rule of combination:

$$m_{1,2}(A) = \frac{1}{1-K} \sum_{B \cap C = A \neq \phi} m_1(B)m_2(C) \tag{4}$$

where

$$K = \sum_{B \cap C = \phi} m_1(B)m_2(C) \tag{5}$$

3.2.2 DS-based Model for Honeypot Detection

Any machine that botmasters intend to compromise could be either a honeypot or a normal machine. Thus, the set of possible propositions is $\mathbb{S} = \{Honeypot, Normal\}$ and the power set is $\mathbb{P} = \{\phi, \{Honeypot\}, \{Normal\}, \mathbb{S}\}$. During the interaction with this machine, including the very first actions such as port scanning, botmasters collect predetermined pieces of evidence from the set of possible evidence $\mathbb{E} = \{E_1, E_2, ...E_n\}$ and determine their BBA values for each possible hypothesis $h \in \mathbb{P}$. BBA values can be determined following different methodologies based on the evidence type. For instance, some evidence can have binary representation of *exist* or *don't exist*, such as detecting virtual environments, which enables the attacker to set the BBA to one of two values only for each hypothesis. Other evidence can have multiple levels of supporting, e.g. penetrating can be very easy, easy, hard, or very hard, which need more detailed BBA values for each possibility. Upon receiving a new evidence, botmaster uses the equation in (4) to combine these pieces of evidence and conclude the degree of belief for this machine. As Dempster rule of combination is associative, the order of evidence is not important and can be combined in any sequence. Although it is possible to calculate the belief of each and every hypothesis, we will only consider calculating it for the proposition 'Honeypot'. After collecting all possible evidence, the final belief of being honeypot is compared with two thresholds, honeypot threshold (h_{th}) and normal machine threshold (n_{th}), where $1 > h_{th} > n_{th} > 0$. The investigated machine is considered as honeypot if the final belief is greater than h_{th}, and a normal machine if less than n_{th}. In other cases, i.e. the final belief is between the two thresholds, the nature of the machine is indeterminable and the botmaster needs to collect more evidence if wants to disclose its true nature. The following is a simplified pseudo code to execute the DS-based detection algorithm:

Example 1 *Consider a case where the botmaster is investigating three machines and has collected three independent evidence from each one (different types of evidence are shown in section 3.3). The botmaster will follow algorithm 1 trying to determine the true nature of each machine. The botmaster starts by initializing the model parameters (* $n = 3, h_{th} = 0.9, n_{th} = 0.1$*) and assigning BBA values for each possible hypothesis of the three machines as in table 2. Here, we directly assign these values for the sake of simplicity, the weight of different evidence will be discussed in section 3.3. After that, the botmaster uses*

Algorithm 1 Calculate Belief of Being Honeypot

```
Require: $n > 1, h_{th}, n_{th} \in [0,1]$
  for $i = 1 \rightarrow n$ do
    Collect Evidence $i$
    Set BBA values for evidence$(i)$
    $i = i + 1$
  end for
  Calculate the belief of being honeypot Bel(H)
  if $Bel(H) \geq h_{th}$ then
    Machine is Honeypot
  else
    if $Bel(H) \leq n_{th}$ then
      Machine is Normal
    else
      Machine type is unknown
    end if
  end if
```

Table 2: BBA values for evidence used in example 1

Machine	*Evidence 1*		*Evidence 2*		*Evidence 3*	
	Honeypot	*Normal*	*Honeypot*	*Normal*	*Honeypot*	*Normal*
A	*0.8*	*0.2*	*0.7*	*0.3*	*0.6*	*0.4*
B	*0.3*	*0.7*	*0.1*	*0.9*	*0.6*	*0.4*
C	*0.6*	*0.4*	*0.6*	*0.4*	*0.4*	*0.6*

equation (4) to combine these pieces of evidence and calculate the belief of being honeypot for each machine. The final values of the honeypot belief of each machine are: A=0.93, B=0.06 and C=0.6. By comparing these values to the assigned thresholds (h_{th} and n_{th}), botmaster can conclude that machine ***A*** *is a honeypot and machine* ***B*** *is a normal machine with a very high confidence. However, machine* ***C*** *true nature cannot be determined based on available evidence and the botmaster needs to collect more evidence to disclose it.*

3.3 Types of Evidence

In this section we present and discuss some examples for different types of evidence that can be collected by botmasters in order to evaluate their belief about the true nature of compromised machines. The presented types of evidence are not the only possible ones that can be used by the botmaster to disclose the true nature of a machine, it is possible to use other types of evidence and/or customize the suggested ones [51] [14] [17] [22] to fit with different types of attacks and different levels of botmasters' abilities. We focus on these types of evidence particularly as they are related to different phases of machine compromising as illustrated next.

3.3.1 Difficulty of Compromising

Computers' operators usually fortify their machines with different defences, such as anti-virus, firewalls and program patches, which make these machines hard to compromise. On the other hand, honeypots, by design, are supposed to appear like easy-to-compromise machines because they aim to attract a large number of attackers [32]. Some interactive honeypots use sophisticated interactions to enforce attackers reveal more of their tools [49], which gives the attacker a feeling that these machines are not easy targets. Nevertheless, the difficulty of compromising is a good indicator of being a honeypot or a normal machine. On the other hand, difficulty is a relative issue, and even in the extreme scenarios, very easy or very hard to compromise, an attacker cannot be certain about the type of the machine. Thus, attacker can assign a belief function that relates the compromising difficulty level to the two possible machine types. Furthermore, attackers can consider different factors in determining the difficulty of exploitation such as:

- Exploited vulnerability: A system that is exploited by using a well-known vulnerability is considered to be an easy-to-compromise. Vulnerability date and the existence of patch for it are important in determining the degree of this factor.

- Number of required trials to penetrate the system: A system that requires more attempts to be penetrated is considered to be harder to compromise and is more likely to be a normal machine rather than a honeypot.

- Number of open ports: A machine with many opened ports is so inviting and is more likely to be a honeypot trap. Honeypot operators would increase the number of opened ports in order to attract more attackers to the trap. Attackers can scan open ports in different ways [14] and they can even use software that is available online for free (for example, Advanced Port Scanner 1.3).

The importance of this evidence comes from the fact that some factors can be measured even before the actual attack (e.g. port scanning) and the possibility to have more than one evidence at the same time, such as considering the exploited vulnerability as one evidence and number of needed trials as another evidence.

3.3.2 Virtualization

It is common, and also preferable, for honeypots to run as virtual machines rather than real ones since this allows the administrators of honeypot to run multiple honeypots on a single physical machine and also makes it easier to control and repair honeypots [38]. Detecting virtual environments [17] can provide botmasters with some evidence about the true nature of the machine since some virtualization features are more likely to be used with honeypots than with normal machines. It should be noted, however, that determining the appropriate type of evidence associated with virtualization is technically involved, especially since many modern applications such as cloud computing [3] have made it common to have normal computer systems running in virtual environments. A rational botmaster considers the targeted environments when determining the value of this evidence. For example, it is more likely to have productive virtual systems for business users rather than for home users. On the other hand, the absence of virtualization can highly support the hypothesis of being a real machine.

3.3.3 Software Diversity

Normal computers typically have different software that are required to perform day-to-day tasks. For example, personal computers usually have MS-Office, OpenOffice or iWork for word processing, spreadsheets and presentations. It is also intuitive to find some Internet browsing software, e.g., Internet Explorer, Chrome, or Firefox, and many other useful utilities. Security-related software is also essential for any real computer system, this includes anti-virus, IDS, and anti-malware. For web servers it is common to have software such as Apache, MySQL and MS IIS. The absence, or lack, of these common software strongly supports the hypothesis of being a honeypot.

Attackers can thus attempt to collect pieces of evidence that support the existence, or absence, of such software and assign them different basic belief values depending on the observed degree of software diversity. The importance of such evidence comes from the easiness of detecting installed software, e.g. listing the contents of '$C : \backslash ProgramFiles$' in Windows systems, which allows for quick determination of the belief degree assigned to such type of evidence.

3.3.4 System Activities

Normal computers are used by legitimate users to perform different tasks. Users may run and terminate programs, create, update and delete files, or surf the web. These activities can be detected in different ways such as monitoring CPU utilization, memory, or hard disk usage. One can also observe running processes by using simple line commands such as '*ps*' in Linux systems. A system that does not show such activities is more likely to be a honeypot rather than a normal machine.

The existence of human actions, such as keyboard strokes, is an indicator that a given machine is being used by legitimate users. Although high interaction honeypots can simulate many of these activities [49], still the absence, or lack, of such activities can be considered as a strong evidence that supports the likelihood of being a honeypot. Botmasters can use many tools to detect system activities and determine the possibility that these activities support the hypothesis of being interacting with a normal or honeypot machine. As normal systems may have some idle time, e.g. off hours in companies or user absence at night time, it is important for the botmaster to measure the targeted system activities during suitable time intervals or by taking the average of these activities over a relatively long period to avoid the effect of idle weekends and night time. Despite the large number of factors that determine the use of this type of evidence, botmasters can find it very important to support the machine type as normal machine when observing a lot of system activities.

3.3.5 Outgoing Traffic

Typically, honeypots do not allow malicious outgoing traffic due to the fear of legal liabilities associated with participation in real attacks [51]. As suggested by Zou and Cunningham [51], botmasters may abuse this fear of legal liability to develop simple test techniques that measure the machine ability to send outgoing traffic. To determine the true nature of a compromised machine, the botmaster may command it to perform what looks like an illicit activity, e.g. spreading malware to other machines, participate in a DDoS attack or send spam emails. However, the targeted victims of these actions are controlled by the botmaster and used as sensors to measure the contribution of the tested machine. Based on the observed response, attackers can convert these measurements into a belief function value that reflects the likelihood of being a normal machine or a honeypot. A key point in here is that honeypots cannot distinguish between real attacks and tests. On the other hand, the lack of response does not necessarily indicate a honeypot. A normal machine may sometimes become unable to respond to such test commands because of a firewall setting, connectivity problems or simply because it is switched off. So attacker may need to repeat such tests and consider each response as a separate evidence that supports being a normal machine or a honeypot with different belief values. Attackers can also collect multiple evidence by performing multi-step test and measure the contribution of the machine in each step separately. For example, to invade a new victims, the bot performs multiple steps such as collecting email address then sending phishing emails with links to websites that contain the malicious code, as in Storm [25] and Waledac [23] bots. Other bots such as Agobot [4] perform port scan to identify possible victims, and then they perform different types of scans that include vulnerability scan, back door scan, and brute force password scan to detect possible entries to the victim. Finally, they send their malicious code through these discovered vulnerabilities. Botmasters can use these steps to delude honeypots and measure their ability to perform different types of illicit actions. The result of each step will be assigned with a belief value of the machine nature. For example, if the botmaster requested the machine to send 100 packets to one target and the sensor reported receiving only 10 of them, this results in higher belief value of being honeypot than the belief value of being a normal machine. On the other hand, if the sensor reported receiving 90 packets out of 100, the belief value of being normal machine will be higher than the one corresponding to being a honeypot. From the attacker perspective, evidence associated with the “Outgoing Traffic" has two advantages. First, it is completely controlled by the botmasters who can decide when and how to make the machine generate such evidence and repeat it as much as needed. Second, the “outgoing traffic” concept is of a special importance as it can indicate the belief of machine type when the bot is unable to send the evidence collected locally in that machine. For instance, suppose the bot has collected some evidence regarding the system activities in a compromised honeypot and was not able to send it back to the botmaster due to the limitation of sending outgoing traffic in the honeypot. In this case, the absence of the outgoing traffic can be considered as an evidence by itself [51]. On the other hand, the other set of evidence makes it easier to disclose low interaction honeypots and have other advantages. For example, the “difficulty of compromising" evidence can be determined without the need of any extra effort, it is somehow a *free* evidence. This evidence does not require the botmaster to issue any special commands through the botnet which reduces the possibility of being detected. Finally, it should be noted that we are not limiting available types of evidence to the above five types. These types are presented as *examples* and it is not hard to imagine other types of evidence that can be collected and analysed during different phases of online attacks [22].

3.4 Numerical Results and Findings

Throughout this section, we assume that attackers are able to collect different pieces of evidence during and after the machine compromising process. Attackers follow the steps described in section 5.2 to update their belief about the nature of the machine they are interacting with. As the Dempster-Shafer rule of combination is associative, it can combine evidence one by one and generate accumulative values until they reach the required level of confidence, i.e. predetermined thresholds, about the nature of the machines under test. Table 3 lists the suggested different machines specifications used in generating the BBA values for evidence combining. Throughout this section, we refer to a high interaction honeypot as ‘H.I.H’ and to a low interaction honeypot as ‘L.I.H’. We assume that the H.I.H is well prepared to mimic the characteristics of a real machine which makes the attacker assigns BBA values similar to what an actual normal machine would have. However, as

Table 3: Specifications of machines

Machine	Compromising	Virtualizing	Software Diversity	System Activities	Outgoing Traffic
Normal	Difficult	No	High	High	Yes
H.I.H	Difficult	No	High	High	No
L.I.H	Easy	Yes	Low	Low	No

determined by [51] the outgoing-traffic test will give results that can distinguish between real machines and honeypots. In what follows, we present our numerical results for different scenarios that show how our suggested DS model can be used by attackers to determine the true nature of the compromised machines.

3.4.1 Distinguishing Normal Machines from Honeypots

In this scenario, we assume that an attacker is trying to determine the true nature of three different types of machines: a normal machine, a low interaction honeypot, and a high interaction honeypot. Table 4 shows the assumed ranges of basic belief values associated with different types of evidence collected by the attacker. In the table, $E1, E2, E3, E4$, and $E5$ represent evidence corresponding to difficulty of compromising, virtualization, software diversity, system activities, and contribution in the botmaster's tests, i.e. outgoing traffic, respectively. As shown in Table 3, we assume that high interaction honeypots are prepared to simulate real machines by providing evidence ($E1 - E4$) in the same way as real machines. As mentioned above, $E5$ is of a special importance to the attackers who rely on the fact that honeypot operators have legal liability constraints that may prevent them from fully participating in real attacks [51]. As shown in Table 3, we assume that honeypots

Table 4: Ranges of BBA for different types of evidence for Different Types of Machines

Machine	E1-E4		E5	
	Honeypot	Normal	Honeypot	Normal
Normal	0-0.1	0.6-0.9	0-0.1	0.6-0.9
H.I.H	0.1-0.3	0.3-0.7	0.6-0.9	0-0.1
L.I.H	0.6-0.9	0-0.1	0.6-0.9	0-0.1

have low contribution percentage that support being honeypot more than supporting being a normal machine, while the high percentage of contribution for the normal machine provides evidence that supports being normal more than it does for being a honeypot. Figure. 3 shows how the attackers' belief about the nature of the compromised machine varies as they collect more evidence. The numerical results were obtained by generating 1000 round scenarios where the BBA values for each evidence is within the assumed ranges in Table 4 with a uniformly distributed probability.

As depicted in Figure 3, attackers are able to detect low interaction honeypots after collecting only few evidence. The need of collecting more evidence becomes apparent when interacting with high interaction honeypots. As shown in Figure 3, the first collected set of evidence supports being a normal machine, which decreases the belief value of being a honeypot. However, after collecting the fifth evidence, the belief of being honeypot starts to increase due to the fact that this evidence is collected by measuring the machine participation in attacker's tests, which eventually leads to determining the true nature of this machine type as a honeypot. From our numerical results, we conclude that if the first few pieces of evidence support being honeypot with high values, then rational attackers would not continue collecting more evidence and is more likely to stop interacting with these machines. However, if the first few pieces of evidence support being a normal machine, then rational attackers are more likely to try collecting more evidence (especially of type $E5$) in order to be sure that they are not being deceived by high interaction honeypots.

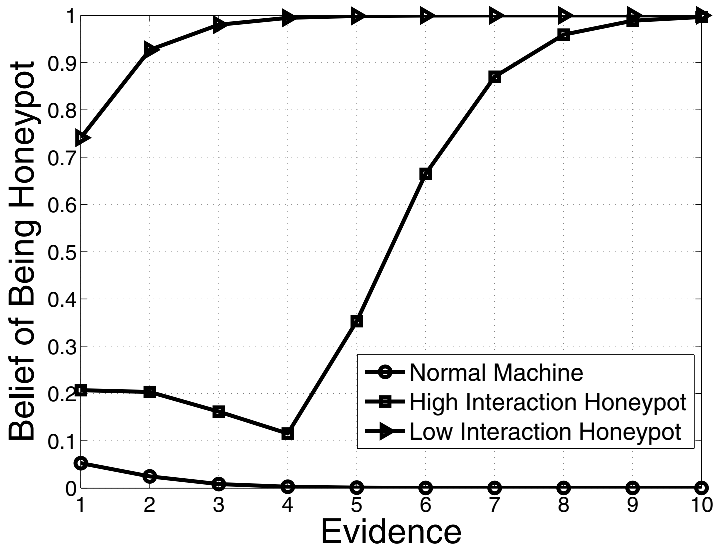


Figure 3: Belief evaluation of being honeypot for normal machine compared with high interaction and low interaction honeypots

DS rule for combining evidence is associative and consequently the order of calculated values does not affect the final belief value. What really affects the value of the final belief is how many pieces of evidence support one hypothesis over the other one and the BBA values for them. Figure 4 shows our numerical results for a scenario that shows how attackers can use the DS model to determine the true nature of three normal machines despite having different BBA values for some evidence. Table 5 shows the values used in obtaining the numerical results that represent the possibility of providing each evidence. In this example, we assume that the machines of type ‘1’ are easy to compromise, machines

Table 5: Ranges of BBA values for different evidence for Three Normal Machines

Machine	Hypothesis	E1	E2	E3	E4	E5
Type 1	Honeypot	0.3-0.7	0-0.1	0-0.1	0-0.1	0-0.1
	Normal	0.1-0.3	0.6-0.9	0.6-0.9	0.6-0.9	0.6-0.9
Type 2	Honeypot	0.3-0.7	0.3-0.7	0-0.1	0-0.1	0-0.1
	Normal	0.1-0.3	0.1-0.3	0.6-0.9	0.6-0.9	0.6-0.9
Type 3	Honeypot	0.3-0.7	0.3-0.7	0.3-0.7	0-0.1	0-0.1
	Normal	0.1-0.3	0.1-0.3	0.1-0.3	0.6-0.9	0.6-0.9

of type ‘2’ are easy to compromise and are deployed as virtual machines, and machines of type ‘3’ are easy to compromise, deployed as virtual machines, and have no software diversity . All machines have a high contribution percentage which supports being normal more than being honeypot. As depicted in Figure 4, by repeating the test multiple times, attackers are able to determine the assumed true nature of the three machines, i.e. normal machines.

Figures 3 and 4 show how attackers can use a single flaw in honeypot design to reveal, and consequently avoid, honeypots. Even with the best equipped honeypot, as in first scenario, repeating the outgoing traffic test would eventually lead the attacker to uncover it. The legal liability of participating in executing botmasters’ commands must have a special attention from the security community to find appropriate solutions to avoid, or reduce, attackers ability of exploiting this particular flaw.

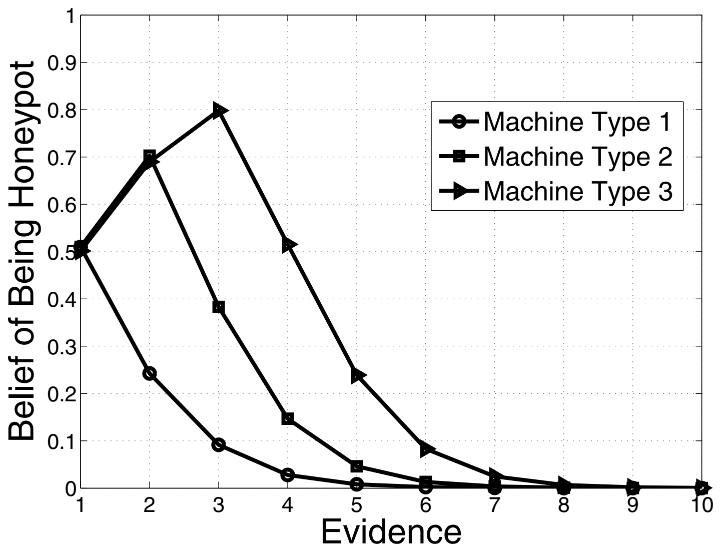


Figure 4: The effect of different supporting evidence values in detecting three normal machines

3.4.2 Using Multi-step Test

It is possible for the attacker to consider several steps as a single test when interacting with a machine by using the outgoing traffic methodology, i.e., by measuring the machine contribution of executing a fake attack test. However, if a normal machine executes only parts of the test but is unable to complete it due to connectivity problems or firewall settings, then it is considered as 'not-contributing', which may make attackers consider it as a honeypot. On the other hand, security professionals can design a high interaction honeypot such that it may contribute in the less severe steps of the attacks, e.g. scanning, without actually doing harmful actions such as infection. To determine the actual contribution of a machine, botmasters can employ DS to combine different evidence that represent the execution of different steps and allocate different BBA values for each step based on its importance and potential legal liability. For example, consider the infection process of Asprox botnet which includes multiple steps [8]. The infection process can be divided into the following three steps:

- Searching for SQL Servers: The bot in the infected machine searches, using Google, for websites that are hosted on Microsoft SQL Server. Botmasters can customize this step by redirecting the search request to one of their sensors that acts as a search engine and measures the execution of the search command.
- SQL Injection: The bot receives a reply from the search engine (sensor) that contains a list of potential victims (other sensors). The bot uses SQL attack tool to attack these sensors, which in turn will measure the contribution of the tested machine.
- Spamming: The bot sends emails to a list of users to lure them to visit the infected website. The botmaster can change the email addresses such that these emails will be received by the sensors to determine the contribution of the machine.

In this part, we compare the contribution of three machine types:

1. Normal machines that highly participate in all steps.
2. Low interaction honeypots that poorly participate all steps.
3. High interaction honeypots that try to avoid detection and also reduce their potential liability by highly participating in the first and last steps of each test (search and spam email) and poorly participate in the second step (SQL injection). The rationale behind these contribution decisions can be justified by the fact that the operators

Table 6: The BBA values for each step in multi-step test

Machine	Hypothesis	Step 1	Step 2	Step 3
Normal	Honeypot	0.2-0.3	0.0-0.1	0.2-0.3
	Normal	0.4-05	0.8-0.9	0.4-05
H.I.H	Honeypot	0.2-0.3	0.6-0.9	0.2-0.3
	Normal	0.4-05	0-0.1	0.4-05
L.I.H	Honeypot	0.6-0.9	0.6-0.9	0.6-0.9
	Normal	0-0.1	0-0.1	0-0.1

of these honeypots can consider the first step as a legal action (searching). Also, while the second step contains a direct attack, the last step can be argued as less jeopardizing, because in most cases the sent email will direct the victim to a safe website due to skipping the SQL injection action in the second step.

Table 6 shows the assumed BBA values associated with evidence collected during these different steps. Botmasters can assign higher BBA values for step two as it represents an illegal action that honeypot operators prefer to avoid due to its legal liability. Figure 5 shows how the botmaster is eventually able to determine the true nature of these three machine types. If a botmaster considered applying 5 tests (that is 15 sub-steps) and setting belief threshold of 0.9 for a machine to be considered as a honeypot and 0.1 for normal machine, it will be easy to recognize the different types of machines.

The numerical results in Figure 5 shows the importance of assigning the BBA following a reasonable approach. All different types of evidence must be evaluated based on several factors depending on its nature, legal liability, the scope of targeted machines and even the attacker experties. For example, an expert attacker may pay less attention for 'Difficulty of Compromising' evidence than what a novice attacker would do. Expert attackers will find

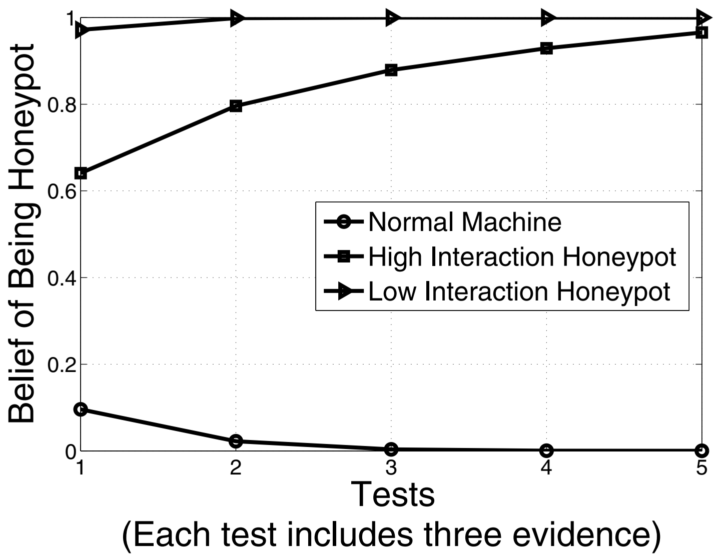


Figure 5: Using multi-step test to determine the true nature of three machine types: normal, low interaction honeypots and high interaction honeypots

more machines as easy targets, and thus the BBA for this evidence should not be very distinctive for them. Another example is when considering the 'System Activities' evidence, it is important to take into consideration the time of test (day/night and weekdays/weekends) and the type of targeted users (home users or business). Some of these considerations might be easy to be automatically performed by the bot itself, e.g. checking system time, while others may need more manual customization by the attackers.

3.5 Chapter Summary

In this chapter, we showed that botmasters can systematically collect, combine, and analyze different evidence throughout and after the machine compromise process in order to determine the true nature of the compromised machines with a relatively high certainty.

Our numerical results show that this technique can be powerful enough to determine the true nature of most compromised machines and honeypots. We hope that the work presented in this chapter would shed some light on weak aspects of current honeypot designs which can be exploited by botmasters to generate and gather evidence that lead to disclosing honeypots and removing them from botnets. Honeypots developers can use this work as a guidance to better enhance the hiding capabilities of their honeypots by making more informed decisions before deciding their response to botmasters commands, e.g., by simulating the current state of the attacker belief and basing their decisions on this estimated belief values.

Chapter 4

A Game Theoretic Investigation for High Interaction Honeypots

In the previous chapter, we showed that botmasters are able to systematically disclose honepyots in their botnets by collecting multiple evidence and combining them in order to calculate their belief about the true nature of the compromised machine. One important evidence to achieve this goal is the machine ability to carry out test attacks, i.e., fake attacks, in which honeypots cannot participate due to the legal liability of participating in illicit actions. This raises the need to analyze the interaction between botmasters and hoenypots in order to determine the optimal percentage of tests that botmasters can send and the optimal percentage of botmasters' commands that honeypots can execute. In this chapter, we present a game theoretical framework for the interaction between botmasters and honeypots. In particular, we model the interaction between honeypots and botmasters as a non-zero-sum noncooperative game with uncertainty. The game solution illustrates the optimal response available for both players. Numerical results are conducted to show the botmasters' behavior update and possible interactions between the game players. The obtained results can be utilized by security professionals to determine their best response to such kind of probes by botmasters.

4.1 Introduction

Honeypots detection methodology suggested by Zou and Cunningham [51] assumes that honeypots do not respond to these kinds of commands by the botmasters. However, if honeypots are designed to completely ignore these commands, then they can be easily detected by the botmasters. On the other hand, full participation by the honeypots in such activities has its associated cost and may lead to legal liabilities [32]. This raises the need for considering the following question: *What is the best response strategy that the honeypot can follow in order to evade detection by botmasters without completely sacrificing the liability?* Furthermore, the execution of such tests by botmasters is bounded to different types of constraints such as firewall settings that might prevent normal bots from always cooperating [46]. Such constraints can lead botmasters to misjudge normal bots as honeypots and drop them from the botnet. To avoid such a false negative decision, botmasters need to run their tests several times in order to build a belief evaluation of the machine nature and then, only when the evaluation belief reaches a specific threshold, botmasters should remove the machine from the botnet. Thus, the question that raises in this context is: *How to differentiate between normal machines and honeypots with a small number of test commands?*

In this chapter, we develop a Bayesian game theoretic framework that models the interactions between honeypots and botmasters in the above setup. In particular, we formulate the interaction between botmasters and the honeypots as a non-cooperative, non-zero-sum game with incomplete information [7] about the honeypot. In this game, botmasters aim to determine their opponents' true nature using the posterior belief function. Botmasters test their opponents by following the methodology suggested by [51]. To make it harder for the honeypots, botmasters mix test commands with real attack commands and repeat the test multiple times.

In summary, our contribution in this chapter is a game theoretic model that:

- Guides security professionals for the best contribution strategy that considers the tradeoff between identity disclosure and legal liability.
- Formally analyzes the moves of botmasters against honeypots.
- Investigates the botmasters' best strategy to differentiate honeypots from normal machines.

4.2 The Game Model

In the this game, we have two players: the botmaster and the honeypot. Throughout the game, the botmaster sends attack commands to the honeypot which responds by participating in the requested attack with a specific contribution percentage (ranging from 0 to 1). Botmasters have incomplete information about the nature of their opponents. In other words, botmasters do not know a-priori the true nature of the machine they are interacting with, i.e., whether it is a normal machine or a honeypot. To disclose the opponent's identity, botmasters command the compromised machines to perform specific actions that can be measured by sensor machines controlled by them. Depending on the nature of the test attack, the botmasters' sensors can use different methods to determine the correctness of the attack as follows:

- Infection Test: In this method, the botmaster commands the bot to infect other machines, where some of these machines act as the attack sensors. When the sensors receive the attack vector, they compare its digest, e.g., MD5 hash, with a previously saved value. If both values are equal, the machine passes the test.
- DDoS Test: The botmaster commands the compromised machine to continuously send web requests to a server that is controlled by the botmaster. From the bot/honeypot perspective, this looks like a DDoS attack. The web server verifies receiving these requests and if the compromised machine sends enough requests, it is considered as a normal bot.
- Spamming Test: The botmaster commands the compromised machine to send emails to group of email addresses which are owned by the botmaster. A machine that sends enough number of such spam emails is considered as a normal bot.

4.2.1 The Players Strategies

The botmaster has three strategies:

1. Test: When choosing this strategy, the botmaster commands the bot to attack the sensor machine using one of the test attacks discussed above. A normal machine would respond to this test by attacking the target machine (the sensor), while a honeypot will have some constraints on fully responding to such commands. To prevent a single point of failure, the botmaster may use multiple sensors which may also increase the accuracy of the test.

2. Attack: In this case, the botmaster commands the bot to attack a real target, for example, by launching a DDoS attack or infecting other machines to recruit more bots.

3. Not Interacting: In this case, the botmaster chooses not to perform any activity with the compromised bot.

The honeypot has two strategies:

1. Contribute: In this case, the honeypot executes the attack commands received from the botmaster. While this helps the honeypot to collect more information about the botmaster and stay for a longer time as a member of the botnet, participating in such illicit activities has its own cost in addition to possible legal liability for damages caused by this attack.

2. Not-Contribute: In this case, the honeypot chooses not to execute the attack commands.

4.2.2 Pay-off

In this subsection, we consider the different factors affecting the players' pay-off.

- Operating cost of the Honeypot/Bot: Even without any interaction between botmasters and honeypots, both have some cost associated with maintaining and running their software and machines. Throughout the rest of the chapter, we use O_h and O_b to denote the operating cost of of the honeypot and the operating cost of the bot, paid by the honeypot operator and by the botmaster, respectively.

- Cost of Attack/Test: When the botmaster interacts with the honeypot, the latter reveals extra information about its true identity, tools and techniques. The revealed information differs depending on the botmaster's strategy, i.e., attack or test. Botmasters need to keep their actual next target as well as their newly developed hacking techniques hidden form security professionals, especially when preparing botnets for a future attack [48]. We use C_a and C_t to denote the cost of performing attacks and the cost of performing tests, respectively.

- Botmaster's Revenue/Loss by Honeypot Contribution: Each bot contribution in a real attack adds more value to it. In case of spamming, each bot has a specific capacity for sending spam emails. The more spam emails sent, the higher the revenue attained by the botmaster. In DDoS attacks, it is important to send enough traffic to the victim server in order to consume its computation and/or bandwidth resources. Typically, a larger number of bots participating in a DDoS attack increases its probability of success. Thus, when the honeypot contributes in a real attack, it adds more value to the botmaster's revenue. If the honeypot does not contribute, this will negatively affect the revenue of the botmaster who initially counts on this participation for the success of the attack. To reflet this payoff, we use R to denote the revenue added to the botmaster's pay-off when the honeypot contributes to real attacks.

- Revealing Honeypots: When a honeypot does not participate in the botmaster's tests, the botmaster has a greater chance of disclosing its true nature and consequently quitting interaction with it. This is considered as an extra reward to botmasters since it improves their business and provides better protection for them. We use E to denote this kind of benefit which is added to the botmaster when the honeypot does not participate in a test attack.

- Liability: When a honeypot participates in a real attack, it acquires the risk of being liable for this contribution. While in some situations taking this risk is prohibitively high, in other scenarios this may not necessarily be the case. For example, when the honeypot is administered in collaboration with law enforcement agencies or when some fees have to be paid to acquire and administer some legal permission in order to be allowed to respond in a very controlled way to such actions. To model the cost associated with this liability constraints, we assign a penalty L for honeypots participating in a real attack.

Table 7 summarizes the payoff factors discussed above.

Having the strategy 'Not-Interacting' means that the botmaster has compromised the machine but does not do anything with it. If this machine is a normal machine, then choosing

Table 7: Summary of players' pay-off

C_a	Cost acquired by the botmaster when performing a real attack
C_t	Cost acquired by the botmaster when performing a test
R	Revenue gained by the botmaster when a machine participates in a real attack
E	Reward to the botmaster when a honeypot does not participate in a test
L	Honeypot liability/penalty for participating in an attack
O_h	The cost of maintaining and running the honeypot
O_b	The cost of maintaining and running the bot

this strategy means that the botmaster spent effort and time for nothing. Furthermore, if this machine is a honeypot, then the botmaster has revealed its identity and the bot code without any further objective. Thus it is intuitive to conclude that the botmaster will not choose such a strategy, and consequently the pay-off table would be reduced to Table 8. Furthermore, we assume that:

- $C_a > C_t$: A real attack contains more important information than a test does. Although some information is common in both types of actions, e.g., source of the command or type of attack, a real attack may reveal the actual target, instead of a fake one, the used code and the vulnerabilities being exploited.

- $L > C_a$: If the liability is less valuable than information obtained from the botmaster, it is clear that a honeypot would always contribute in executing botmasters' commands to stay in their botnets and get more information. However, as argued in [51], this is not usually the case.

- $R > C_a$: Botmasters are financially motivated [18]. If the revenue obtained from a bot does not exceed the risk of revealing the attackers' information when interacting

Table 8: Pay-off table when the botmaster interacts with a honeypot

Strategy	Contribute	Not-Contribute
Test	$-C_t - O_b, C_t - O_h$	$E - C_t - O_b, C_t - E - O_h$
Attack	$R - C_a - O_b, C_a - L - O_h$	$-C_a - O_b, C_a - O_h$

Table 9: Pay-off table when the botmaster interacts with a real bot

Strategy	Contribute	Not-Contribute
Attack	-O_b+R, 0	-O_b,0

with it, then it would not be profitable for the botmaster to recruit this bot.

Under the above reasonable assumptions, it is straightforward to show that there is no pure strategy Nash equilibrium for this game. In what follows, we use a mixed strategy to solve this game. Let p denote the probability that the botmaster chooses to launch a test and q denote the probability of the honeypot chooses to participate in executing the botmaster's commands.

4.2.3 Utility Function

The utility function of botmasters is affected by the uncertainty of their opponents. Botmasters use the posterior belief function, $\mu_t(\theta_h)$, in calculating the pay-off associated with their strategies where $\mu_t(\theta_h)$ reflects the botmasters' belief about the type of the machine that they interact with, i.e., whether it is a honeypot or a normal machine. When $\mu_t(\theta_h)$ approaches 1, the botmaster becomes certain that the machine is a honeypot. At the beginning of the evaluation process, botmasters set $\mu_0(\theta_h)$ to reflect their a-priori belief about the nature of the machine. In other words, $\mu_0(\theta_h)$ is set to the value that reflects the botmaster's belief about the percentage of honeypots in the botnet. A high value (close to 1) for $\mu_0(\theta_h)$ can be used by a suspicious botmaster who may have some reasons to believe that the botnet is penetrated by a large percentage of honeypots. Conversely, setting $\mu_0(\theta_h)$ close to 0 implies that the botmaster is almost certain that the botnet is not infiltrated by any honeypots. Then the botmaster updates its belief as follows:

$$\mu_{t+1}(\theta|a) \quad = \quad \frac{\mu_t(\theta_h) \times P(a|\theta_h)}{\mu_t(\theta_h)P(a|\theta_h) + (1 - \mu_t(\theta_h))P(a|\theta_n)} \tag{6}$$

where $\mu_{t+1}(\theta_h|a)$ denotes the belief of having the opponent type as 'Honeypot' at time $(t+1)$ after observing the strategy a, $\mu_t(\theta_h)$ denotes the value of the belief function at time t, and $P(a|\theta_h)$, $P(a|\theta_n)$ denotes the probability of observing strategy a from a honeypot and from a normal machine, respectively. Let $P_t(a = Contribute|\theta_n) = U$. One can find a good estimate for U by evaluating the expected percentage of working time for a normal machine, e.g., we may assume that $U = \frac{h}{24} \times \frac{d}{7}$, where h denotes the average number of working hours in a day and d denotes the average working days in a week. U can also be affected by other factors such as firewall settings, temporarily failures of normal machines, and geographical and time zones distribution of the botnet.

After updating the belief function, the botmaster calculates the utility function as:

$$\begin{aligned}U_a &= \mu(\theta_h)[-pq(C_t+O_b)+p(1-q)(E-C_t-O_b)\\&\quad +(1-p)q(R-C_a-O_b)+(1-p)(1-q)\\&\quad (-C_a-O_b)]+(1-\mu(\theta_h)[U(R-O_b)+(1-U)(-O_b)]\\&= -\mu(\theta_h)UR+UR-\mu(\theta_h)C_a+\mu(\theta_h)pC_a-O_b\\&\quad +\mu pE-\mu(\theta_h)pqE-\mu(\theta_h)pqR-\mu(\theta_h)pC_t+\mu(\theta_h)qR\end{aligned}$$

Let q^* denote the optimum value of q, i.e., the optimum participation percentage by the honeypot to attack commands received from the botmaster, i.e., we have

$$U_a(p^*,q^*) \geq U_a(p,q^*).$$

We can determine q^* by calculating the first derivative of U_a with respect to p and equating it to zero. Thus we have

$$\mu C_a+\mu E-\mu q^*E-\mu q^*R-\mu C_t=0 \Rightarrow$$

$$q^*=\frac{E+C_a-C_t}{E+R} \tag{7}$$

Similarly, the utility of the honeypot is given by

$$\begin{aligned}U_h &= pq(C_t - O_h) + p(1-q)(C_t - (E + O_h)) \\ &\quad +(1-p)q(C_a - (L + O_h)) + (1-p)(1-q)(C_a - O_h) \\ &= C_a - pC_a - pE + pqE + pqL - qL - O_h + pC_t\end{aligned}$$

Let p^* denote the optimum value of p, i.e.,

$$U_h(p^*, q^*) \geq U_h(p^*, q)$$

Similar to the above analysis, we determine p^* by calculating the first derivative of U_h with respect to q and setting it to zero

$$p^*E + pL - L = 0 \Rightarrow p^* = \frac{L}{E+L} \tag{8}$$

Thus to maximize their utility, botmasters send attack commands with probability

$$\begin{aligned}P_a &= \mu(t) \times (1-p^*) + (1-\mu(t)) \times 1 \\ &= 1 - \mu(t) \times p^*\end{aligned} \tag{9}$$

and send test commands with probability

$$P_t = \mu(t) \times p^* \tag{10}$$

4.3 Game Discussion

From the game model above, it is clear that botmasters will eventually be able to disclose the true nature of their opponent by observing the opponent's responses and consequently updating their belief function about it. Botmasters are also able to optimize their utility by adjusting the percentage of test commands relative to actual attack commands using equation (9) after updating their belief value. On the other hand, a honeypot is not able to determine the true nature of the botmaster's commands, i.e., whether the commands correspond to actual attacks targeting real victims or tests targeting sensors controlled by the botmaster. Consequently, the honeypot cannot update its best response strategy, q^*, over the time which indicates an inherent deficiency in the current honeynets' design. In what follows, we further discuss the impact of the parameters (E, L and R) used to calculate the optimal response strategy for both players.

- For situations where the honeypot penalty for participating in an attack is very high compared to the reward of the botmaster gained by discovering the honeypot true nature, e.g., in situations where legal liability is excessive, we have

  $$p^* \approx 1, P_a \approx 1 - \mu(t), \text{ and } P_t \approx \mu(t).$$

  Thus, in these situations, botmasters will use the belief value $\mu(t)$ to determine the percentage of test and attack commands. Higher values of the belief function lead to more test commands and less attack commands. In this case, the interaction between the botmaster and honeypot can be seen as an accelerated loop where the honeypot does not respond to tests and thus the belief value increases, which means that more tests are sent. This will significantly expedites the honeypot detection process.

- If the penalty of the honeypot for participating in real attacks is much less than the botmaster's reward for disclosing the honeypot, i.e., $L \ll E$, then p^* will assume a small value and the botmaster will tend to send less tests compared to the previous case. This is because the honeypot is expected to participate more in executing the botmaster's commands, which enables the botmaster to make use of this good-participating machine in executing real attacks rather than spending time and effort on sending many tests to disclose its true nature.

To simplify our analysis, throughout the rest of this section, we assume that $C_a \approx C_t$ and hence we have $q^* \approx \frac{E}{E+R}$. In general, q^* is effected by both the botnet size and honeypot penetration percentage. In large botnets, the revenue obtained by the botmaster when a given machine participates in an attack, R, would typically be small. Also, the reward E of removing a honeypot from botnet increases if the number of honeypot is small, e.g., if the botnet is infiltrated with only one honeypot then removing this honeypot will prevent all the information leakage from this botnet.

- For $E \ll R$, e.g., in the case of a small botnet that is highly infiltrated with honeypots, q^* will have a small value, $q^* \ll 1$. In other words, honeypots will not participate in executing the botmasters' commands most of the time because there are so many honeypots gathering information about the botmaster and the botnet. Consequently, discovering and removing one of these honeypots does not have a big impact on the honeynet performance and hence honeypots tend to avoid the penalty of participating in real attacks by reducing their contribution percentages.

- For $E \gg R$, e.g., in the case of a large botnet with few honeypots, the percentage of contribution increases because security professionals need to keep their honeypots in the botnet.

4.4 Numerical Results

In this section, we present our numerical results that show the effect of different parameters on the temporal variation of the belief function of the botmaster as well as on the optimal probability of sending real attack commands by a rational botmaster.

4.4.1 Belief Evaluation

Assuming that a normal machine responds to attack/test commands only 5 days/week, 8 hours/day and ignoring other temporarily networking problems such as possible firewall blocking, then $U = \frac{8}{24} \times \frac{5}{7} \approx 0.23$. Figure 6 shows the variation of $\mu_t(\theta_h)$ for different contribution percentages by the honeypot for botnets with different infiltration percentage (from low to high) as reflected by the botmaster's initial value of the belief function $\mu_0(\theta) = 0.01$, 0.1 and 0.5 and 0.9, respectively. As depicted in Figure 6, the value of the initial belief does not have a big impact on the evolution trend of the botmaster's belief. It is also clear that when the honeypot does not respond to the botmaster's commands, or responds to it in small percentage of times less than U, then the botmaster is able to detect its true identity

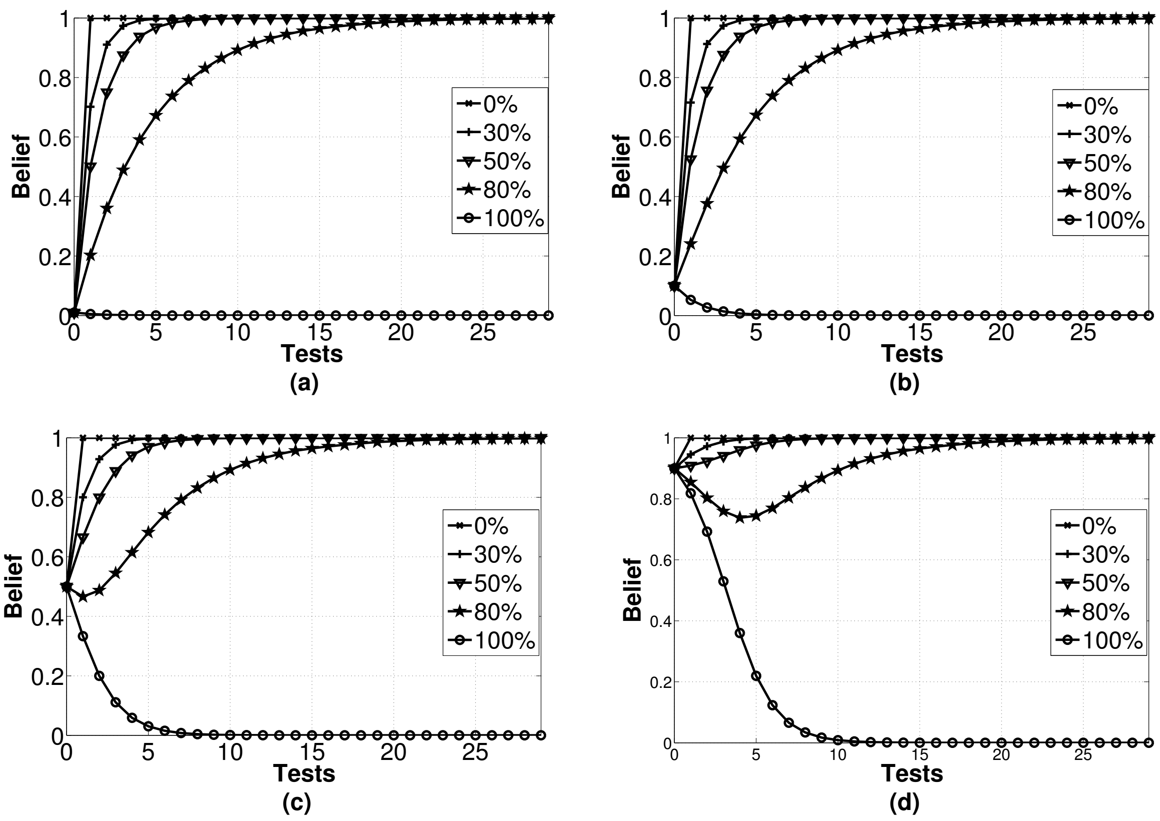


Figure 6: Botmaster belief for $\mu_0(\theta_h) = 0.01, 0.1, 0.5, 0.9$ and $U = 0.23$

using a relatively small number of tests. On the other hand, it is interesting to note that increasing the participation percentage of the honeypot in the attacks is not always a good strategy to avoid detection by the botmaster. In particular, when the honeynet responds to the attack/test commands by probability much greater than U, then the botmaster is also able to easily detect its true nature. In other words, to avoid detection by the botmaster, what is really important is to mimic the behavior of a normal machine.

From the above numerical results, we can conclude the following:

- To prolong its stay in a botnet, the honeypot should respond with probability $U \pm \epsilon$ for small values of ϵ. Furthermore, in order to improve its utility, the honeypot is better to respond with $U - \epsilon$.

- It is easier for the honeypot to evade detection in botnets where U has a small value, e.g., when the compromised machines are in different geographical locations and thus are not presumably all active on the same time.

4.4.2 The Impact of Changing the Honeypot Sequence of Actions

In this section, we study the impact of different distributions of honeypot actions (contribute and not-contribute) for the same contribution percentage. We consider a 50% contribution percentage for the honeypot, i.e., the honeypot is assumed to participate in half of the commands received from the botmaster. We also assume that $U = 0.7$, i.e., a normal machine is expected to respond to 70% of the botmaster commands. We compare the effect of different sequence of actions for the honeypot by counting the number of tests required to reach a specific belief value threshold (a threshold belief value of 0.8 is be used in the following illustrating example).

Figure 7 shows the honeypot response as a series of ones (Contribute) and zeros (Not Contribute). Under our assumptions, the best strategy for the honeypot is to make all of its contributions at the beginning then to stop contributing as shown in Figure 7-(a). However, this is not possible in practice since the honeypot cannot predict how many tests the botmaster will perform. Figures 7-(b) and 7-(c) show similar sequence of actions where the honeypot starts with contribute in (b) and with not-contribute in (c). The belief value of 0.8 is reached after three tests only in (b) and fifteen tests in (c).

The best *realistic* contribution decision in this scenario is in Figure 7-(d) '111000111000...'

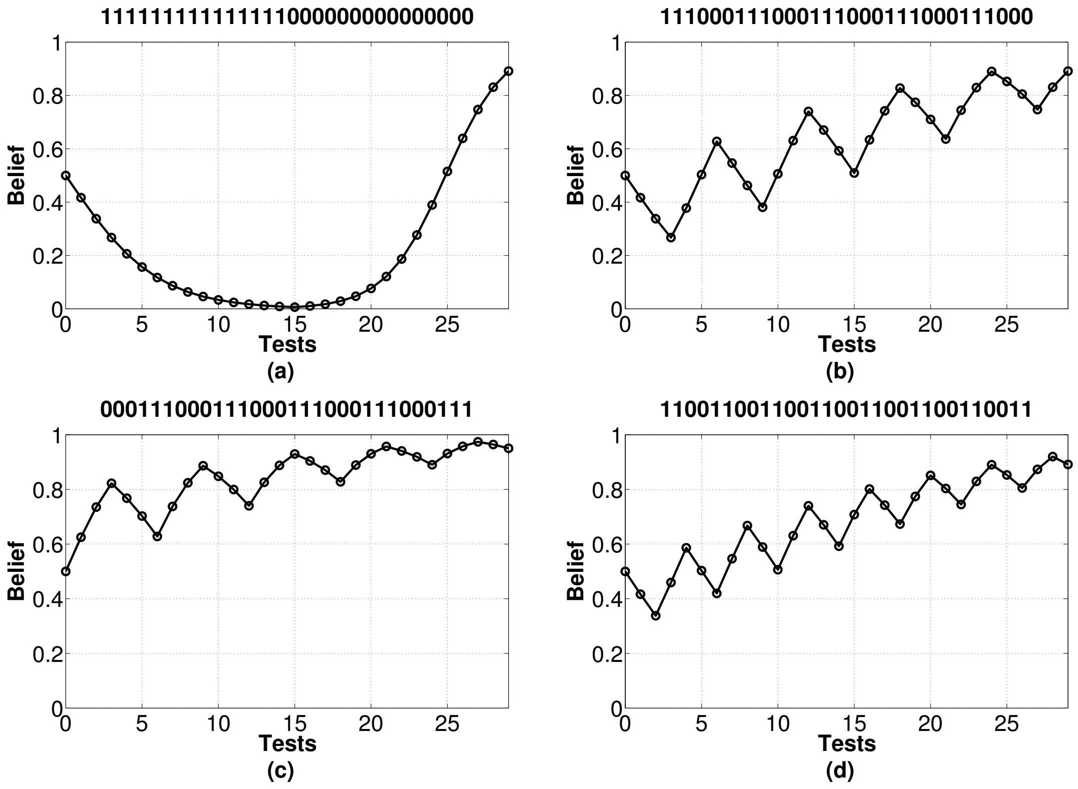


Figure 7: Belief evaluation with different contribution schemes

where the belief value reaches 0.8 after eighteen tests. These results shows that:

- Irrespective of the honeypot sequence of actions, the botmaster will eventually evaluate the same value of the belief function as long as the same percentage of contribution is used by the honeypot. The only effect of changing sequence of actions is to delay reaching a higher belief value, which may encourage the botmaster to send more real attack commands to the honeypot for a longer time.

- Honeynets need to enhance their hiding capabilities by making intelligent decisions before deciding their response to botmasters' commands. For example, if the target of the botmaster command is a popular reputable server, then this command would most likely correspond to a real attack. Consequently, future honeypots should be designed to achieve a better decision by inquiring some side channel information about the target of the botmaster command such as geographical location and IP reputation (e.g., by querying online services such as *WatchGuard ReputationAuthority* available at www.reputationauthority.org).

4.4.3 Honeypot and Botmaster Interaction

In this section we explain how botmasters respond to honeypot actions by updating their belief, probability of attack, P_a, and sequence of actions (test/attack). We present the botmaster actions by a '0' for a test and a '1' for a real attack. Similarly, the honeypot actions will be denoted by '0' for not-contribute and '1' for contribute. We consider the following factors in analyzing the interaction between the botmaster and the honeypot:

- Percentage of participation in tests: This is important because it affects the calculation of the belief value.
- Liability: This is calculated based on percentage of participation in each step of real attack commands.
- Multi-step attack success: A single-step attack [12] denote the case where the attack is executed by one command only (e.g. in a spamming attack, send emails to *list-of-victims@domain.com*). On the other hand, multi-step attacks [12, 31] require the botmaster to send more than one command to accomplish it. The success of a multi-step attack requires the honeypot to contribute in all steps of the attack.

At the beginning, the botmaster and the honeypot calculate their optimal strategies. The honeypot calculates the percentage of contribution q^* from equation (7) and the botmaster calculates the percentage of real attack commands using equation (9). To simplify our analysis, we take $E = L = R, C_a = C_t$, $U = 0.7$ and $\mu(0) = 0.5$. This makes $q^* = 0.5$, i.e., the honeypot contributes in half of the botmaster's commands, and $P_a = 0.75$, i.e. the botmaster probability to send real attack commands is 75%. The honeypot makes its decision at the beginning of the interaction and cannot optimize its response strategy with time because it cannot tell the true nature of the received botmaster commands and it also cannot predict how the botmaster changes its belief function based on its own response.

Consequently, while the honeypot randomly distributes its actions using the fixed value of q^*, the botmaster distributes its actions randomly considering the value of P_a and P_t, which are updated after each test based on the new belief value μ_t.

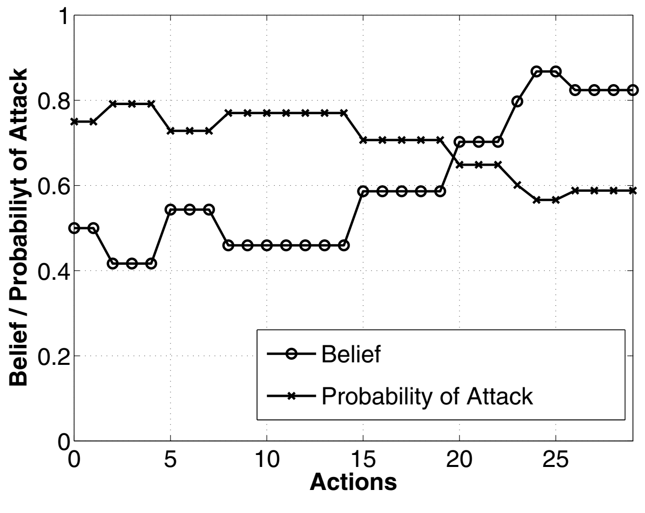


Figure 8: Belief and probability of attack evaluation for the parameters assumed in section 4.4.3

Table 10: Honeypot / botmaster Interaction

Honeypot	111000111001001000110110011100
botmaster	110110110111111011110110010111
Test Participation	3/5 (60%)
One-Step Attack Success	12/22 (54.5%)
Three-Step Attack Success	1/7 (14%)

In the following example, we analyze the interaction between the honetpot and the botmaster using thirty sequence of actions: (111000111001001000110110011100) for the honeypot responses and (110110110111111011110110010111) for the botmaster actions. Figure 8 shows the changes of the belief value $\mu_t(\theta_h)$ and the probability of attack Pa according to honeypot's responses to tests. It is clear that the honeypot response to a real attack command does not change the belief value nor the attack percentage since, in this case, the botmaster is not able to check the bot response. According to the calculated values of P_a, table 10 shows the series of actions for botmaster and honeypot, and the percentage of honeypot's participation in tests, the percentage of honeypot attack success rate for single-step attack, and attack success rate for three-steps attack. The above results also show how the liability is a crucial consideration for security professionals. Even with good percentage of contribution in tests, the chance of being part of a real attack, and consequently the chance of being liable, is high. On the other hand, while the chance of getting involved in all steps of a multi-step attack is relatively smaller, the liability resulting from participating in a successful attack (14% in the above example) is still considerable.

4.5 Chapter Summary

In this chapter, we presented a game theoretic framework to analyze the interaction between security professionals, who are trying to infiltrate botnets using honeypots, and botmasters who are trying to detect the presence of honeypots in their botnets. The results obtained from our analysis allow the security professional to optimally decide on the best response to probes sent by the botmasters in order to disclose their honeypots. Our analysis also shows some inherent weakness in the current design of honeypots; since botmasters are able to probe the honeypots with test/attack commands, botmasters are able to update their belief about the machines they are interacting with and consequently optimally decide on the optimal mix between real attack commands and test attack commands. On the other hand, honeypots' decisions cannot be systematically optimized over the time because the true nature of the botmaster probes are not known to them.

Chapter 5

Markov Decision Processes Model for High Interaction Honeypot

In this chapter, 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 numerical results that show the honeypots optimal response strategies and their expected rewards under different attack scenarios.

5.1 Introduction

As we showed in the previous chapters, the current honeypot technology is not capable of avoiding detection techniques that exploit the legal liability, such as the one suggested in [51]. This raises the need for deciding the optimal strategy that prolongs honeypots lifetime in the botnet while avoiding high legal liability. In this chapter, we model the interaction between the honeypot and the botmaster using Markov Decision Processes (MDPs) [39] in order to determine the honeypot optimal response to such test techniques. In particular, the honeypot-botmaster interaction is modeled by MDPs with a set of states, strategies, 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 strategy, 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, hoenypot rest cost, probability of attacks and probability of disclosure. Then we extend our model to a more realistic scenario using Partially Observable Markov Decision Processes (POMDPs) 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 operators of the honeypot. The two models presented in this chapter can help security professionals to:

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

5.2 Markov Decision Process Model for High Interaction Honeypot

In this section, we briefly review the principles of Markov Decision Processeses (MDPs), and present our developed MDPs model for honeypot interaction with botmasters. A Markov Decision Process [36] is a tuple (S, A, P, r, d), where S represents the set of system states, A represents the set of possible strategies, 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 strategy a. The reward function r represents the gain for using strategy 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.

5.2.1 States of Honeypot

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 is 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 [16] as it is possible that botmasters focus on expanding their botnets and do not communicate intensively with existing botnet members.

5.2.2 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 Strategies: At each state, the honeypot can choose one of the following three strategies:

    - 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 strategy 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 strategy is used to prevent the attackers from sending malicious traffic from the honeypot. One reason for following this strategy 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 [49]. 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 strategies 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 (strategy 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 botmsters 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.

5.2.3 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. C_I (Information gain): 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 commands of botmasters 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.
$C_O < C_I$ and $C_R < C_I$; 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 9, 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 (strategy, reward, probability). For example, (R,$-C_I$,1) means that the system transits with probability 1 when strategy R is used and the associated cost is $-C_I$. 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 (strategy A) or not (strategy 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 (strategy A) which makes the botmasters think that the honeypot is a real machine. Consequently, more information (C_I) 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 (strategy 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 (C_I). If the honeypot is disclosed, it moves to state D and stays there until it is reset to the starting state W (strategy 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 strategy comes with the cost of C_R. Figure 9 depicts the developed MDPs model.

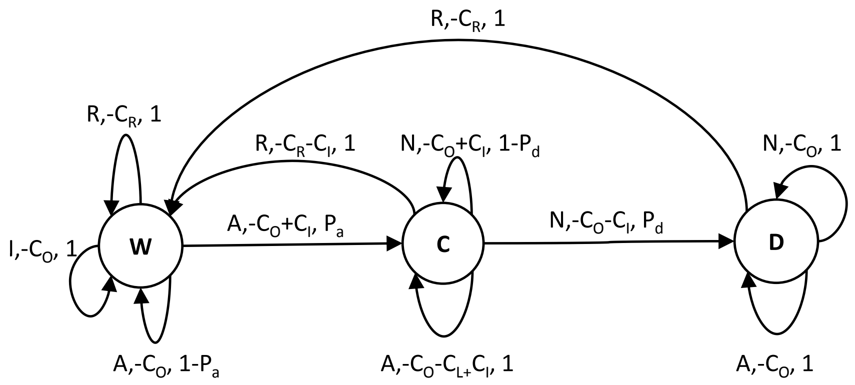


Figure 9: The developed MDPs model for the honeypot interaction with botmasters.

5.2.4 Analysis of the Developed Model

A recurrent MDPs 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 MDPs model [43]. 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 [43], considering a discount factor $d = 1$.

Let *N* be the number of states in a MDP and let *P* denote the matrix of transitions probabilities

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

where p_{ij} denotes the probability of transiting from state *i* to state *j*.

The vector of steady state probabilities is given by

$$\pi = [\pi_1 \quad \pi_2 \quad \dots \quad \pi_N] \tag{12}$$

where

$$\sum_{i=1}^{N} \pi_i = 1 \tag{13}$$

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

$$\pi = \pi \times P \tag{14}$$

with consideration of equation (13). Let *R* denote the matrix of rewards

$$R = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1N} \\ r_{21} & r_{22} & \dots & r_{2N} \\ \dots & & & \\ r_{N1} & r_{N2} & \dots & r_{NN} \end{bmatrix} \tag{15}$$

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

$$q_i = \sum_{j=1}^{N} p_{ij} r_{ij} \quad , \quad i = 1, 2, ..., N \tag{16}$$

The gain is calculated as

$$G = \sum_{i=1}^{N} \pi_i q_i \tag{17}$$

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.

5.3 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 strategies 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 strategy 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 strategy N or R it will not be able to join the botnet and cannot provide security professionals with useful information. So the only logical strategy at state W is to allow the botmasters to compromise the honeypot (strategy A). Thus, we are left with only three policies to choose from, i.e., policies in which the honeypot always chooses strategy A at state W and strategy R at state D. We denote these policies by the name of the strategy used at state C:

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

To decide the optimal policy for the honeypot, we calculate the gain obtained in each policy using equation (17) 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*:

$$G_A = C_I - C_O - C_L \tag{18}$$

   The gain obtained by applying this policy can be positive only if $C_I > C_O + C_L$. 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*:

$$G_N = \frac{(P_a - P_a P_d)C_I - (P_a + P_d)C_O - P_a P_d C_R}{P_a + P_d + P_a P_d} \tag{19}$$

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*:

$$G_R = \frac{-C_O - P_a C_R}{1 + P_a} \tag{20}$$

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 parameters of the system, 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 ($C_O = 1, C_R = 2, C_I = 20, C_L = 15$ and $P_a = 0.5$). Figure 10 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 strategy 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.

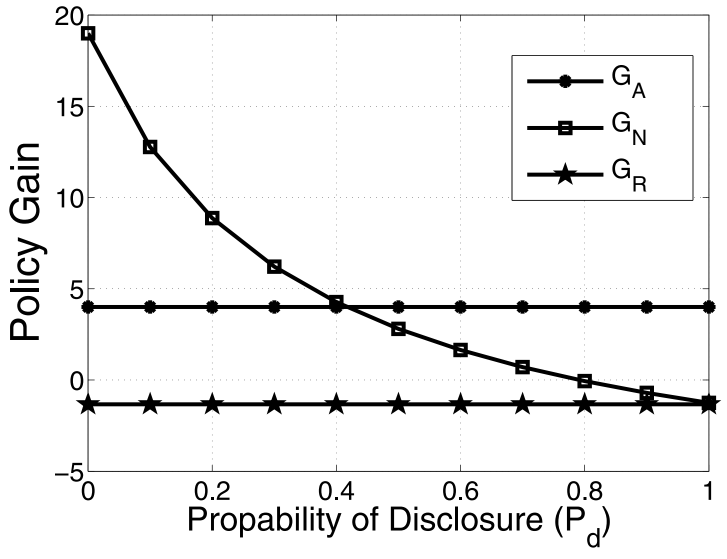


Figure 10: Changes in the gains (G_A, G_N, and G_R) as a function of probability of disclosure (P_d).

5.4 Modeling Uncertainty of the Honeypot State

When using MDPs, 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 strategy 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 strategies. 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 MDPs. 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) [10], 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.

5.4.1 Partially Observable Markov Decision Processes

The definition of POMDPs is similar to MDPs with the addition of two parameters:

1. A finite set of observations $\mathcal{O}$.
2. An observation function

   $\boldsymbol{O}$: $\boldsymbol{A} \times \boldsymbol{S} \mapsto [Pr(O_1), Pr(O_2), ...], \sum_i (Pr(O_i) = 1$

In POMDPs, after executing strategy 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 [10]. However, this solution contradicts with the definition of MDPs in two aspects:

1. Maintaining The Belief State:

   The entire history of strategies 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 strategy. However, as described in [10], 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:

   $$b(s)_{t+1} = \frac{Pr(o|s, b_t, a) \times Pr(s|b_t, a)}{Pr(o|b_t, a)} \tag{21}$$

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

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 [10]. To overcome this problem, approximation algorithms [9] [28] are used to find an approximation to the optimal solution. All algorithms use value iteration to calculate the approximated solution [10] starting at an initial value function *V(b)* for the initial belief state, and then iterating using equation (22) as follows:

$$V'(b) = max_a[\sum_s b[s]R[s,a] + \Upsilon \sum_o Pr(o|b,a)V(b')] \tag{22}$$

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 POMDPs reside in the way they sample the belief space to reach the optimal solution [9] [28].

5.4.2 The POMDPs 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 [32]. 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 $Commands$.

5.5 Numerical Results

In this section, we present and discuss our numerical 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) [15] 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 [28] 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 [28].

5.5.1 Scenario of High Liability Cost

In what follows, we assume the following parameters values:

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 $C_I = 10$, 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 $Pr(Absence) = 1.0$.

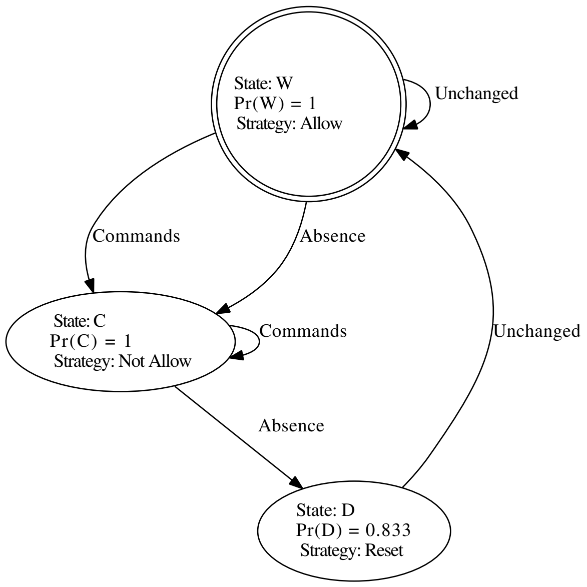


Figure 11: An example for a honeypot optimal policy graph representation in a high liability scenario $(P_a = 0.7, P_r = 0.6)$.

Figure 11 shows the optimal policy for the assumed set of parameters. The system starts in state W with probability 1 and chooses the optimal strategy A to allow botmasters to compromise the heoneypot. If the honeypot is compromised, the system will certainly transit to state C with probability 1, in which the optimal strategy 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 strategy is to reset the honeypot to its initial state W. To summarize, in this scenario, the optimal policy for the honeypot is to use strategy A when observing $Unchanged$, to use strategy N when moving from state A (regardless of the observations) and to use strategy R when having the observation $Absence$ after using the strategy N.

5.5.2 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 parameters value changes 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 12 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, which increases the value of C_I in our model. 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 stay for a longer time 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.

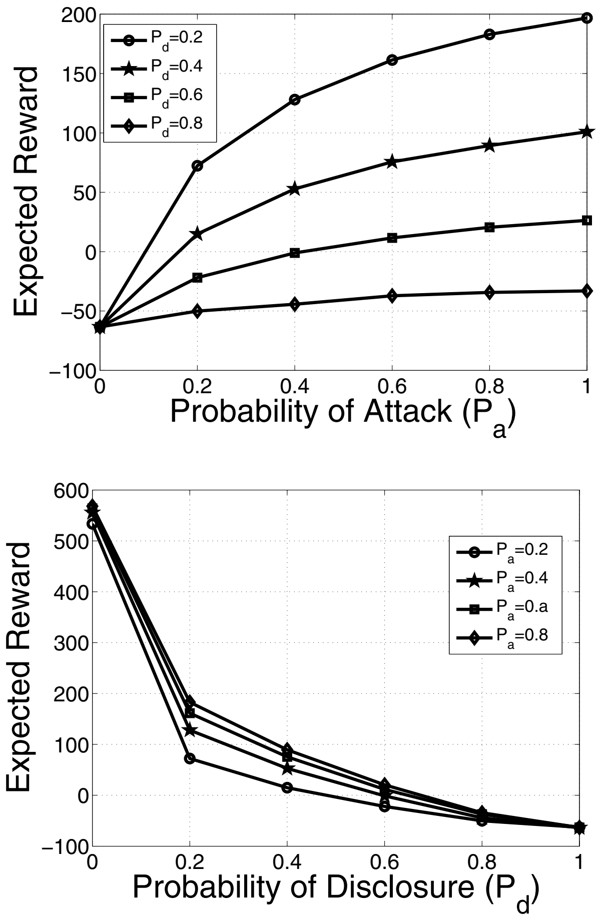
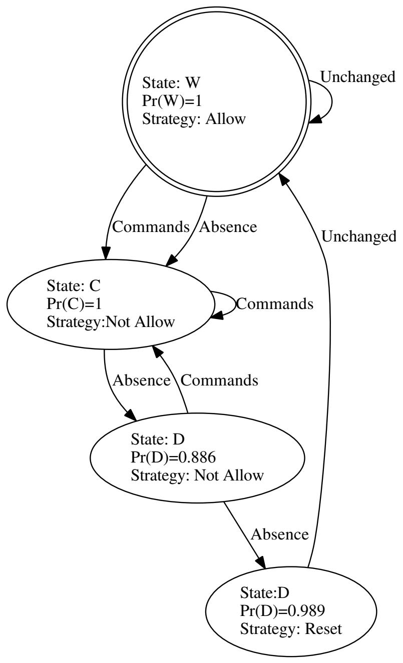
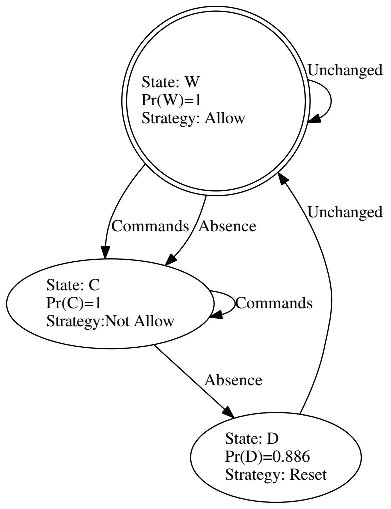


Figure 12: The system expected reward increases with higher probability of attack, P_a, and decreases with higher probability of disclosure, P_d.

Figure 13 shows the optimal policies for two sets of parameters that differ only in the value of probability of attack (in Figure 13-a, P_a=0.1 and in Figure 13-b, P_a=0.5). As depicted in Figure 13-a, due to the low value of P_a, the system is trying to capture more



(a) P_a=0.1



(b) P_a=0.5

Figure 13: Examples for the optimal policy graph for different values of probability of attack, P_a.

information by choosing the strategy 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 strategy R and reset to the initial state W. For $P_a = 0.5$, in Figure 13-b, the system directly chooses the strategy 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 rest the system and wait for new attacks rather than waiting for the current attackers to make new interactions with the honeypot after observing the absence of their commands.

5.6 Chapter Summary

In this chapter, 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 legal liability of the honeypots 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.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we analysed the interaction between botmasters and honeypots using different mathematical tools. In particular, we have:

- Introduced a game theoretical framework that enables security professionals to decide their best strategy against probes, performed by botmasters, which exploit the legal liability. Our model shows that botmasters can systematically build up a belief value of the honeypot true nature which leads eventually to disclose the honeypot and consequently remove it from the botnet. We have calculated the optimal percentage at which security professionals should allow honeypots to participate in botmasters' probes. This optimal contribution strategy enables the honeypot to provide the best outcome against such probing techniques.
- Proposed a technique that enables botmasters to systematically collect and analyze

evidence throughout the compromising process of a victim machine. These pieces of evidence enable botmasters to evaluate the true nature of the machine by using Dempster-Shafer theory of evidence combining. Botmasters can assign a basic belief value for each piece of evidence, which reflects how much the evidence supports the proposition of being a honeypot (or the proposition of being a normal machine) and them use Dempster rule of combination to combine these evidence and calculate the final belief value about the true nature of the machine. We also investigated different types of evidence that are related to different phases of the compromising process and presented several scenarios to show how botmasters can detect the true nature of the investigated machine.

- Modelled the interaction between botmasters and honeypots by a Markov Decision Process (MDP) and then determined the honeypots optimal policy for responding to the commands of botmasters. The model was 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 and consequently prolong the honeypot lifetime inside the botnets. We have also provided numerical results that show the honeypots optimal response strategies and their expected rewards under different attack scenarios.

6.2 Future Work

Our research can be extended in the following directions:

- Developing Better Honeypots: Throughout our research, we have noticed that the current honeypot technology has inherit limitations that enable botmasters to detect honeypots and remove them from their botnets. Honeypots must be able to automatically distinguish actual attacks from fake ones in order to avoid such detection techniques. Thus, honeypots need to make intelligent decision about executing the commands of the botmasters. One possibility is to make use of the reputation of the target; if the target of the botmaster command is a popular reputable server, then this command would most likely correspond to a real attack. On the other hand, if the target of the command has already been blacklisted, then most likely this is a test command and the honeypot should execute it. Consequently, future honeypots should be designed to achieve a better decision by inquiring some side channel information about the target of the botmaster command such as geographical location and IP reputation (e.g., by querying online services such as *WatchGuard ReputationAuthority* available at www.reputationauthority.org).

- Empirical Experiments: The developed mathematical techniques, such as Dempster-Shafer, can be embedded into experimental research bots to measure its ability to distinguish honeypots from real machines. It is also important to test the effects of such implementation on the bot itself, e.g., overhead resulting from extra tasks. The obtained experimental results would allow us to better judge the effectiveness of the use of false evidence as a defence strategy.

- Enhancing The Markov Model: We have developed and analyzed our Markov model under simplified assumptions. The model can be enhanced to include estimation for the values of different parameters, especially, the probability of attack P_a and the probability of disclosure P_d. The values of such parameters can be estimated experimentally by exposing honeypots to the wild. On the other hand, we have assumed infinite number of interactions between the botmasters and the honeypots in our Markov model. A more precise model that takes into account the finite number of interactions between the botmasters and the honeypots can be developed.

Bibliography

[1] Z.H. Abbas and F.Y. Li. Energy optimization in cellular networks with micro-/pico-cells using Markov decision process. *18th European Wireless Conference, EW 2012*, pages 1–7, 2012.

[2] M. Akiyama, T. Kawamoto, M. Shimamura, T. Yokoyama, Y. Kadobayashi, and S. Yamaguchi. A proposal of metrics for botnet detection based on its cooperative behavior. In *International Symposium on Applications and the Internet Workshops*, pages 82–85, 2007.

[3] M. Armbrust, A. Fox, R. Griffith, Joseph A., R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. In *Technical Report No. UCB/EECS-2009-28*. University of California at Berkley, USA, 2009.

[4] P. Barford and V. Yegneswaran. An inside look at botnets. In *Malware Detection*, volume 27 of *Advances in Information Security*, pages 171–191. Springer US, 2007.

[5] J.O. Berger. *Statistical decision theory and Bayesian analysis*. Springer, 1985.

[6] J. Bethencourt, J. Franklin, and M. Vernon. Mapping internet sensors with probe response attacks. In *USENIX Security Symposium*, pages 193–208, 2005.

[7] K. Binmore. *Game Theory A Very Short Introduction*. Oxford University Press, 2007.

[8] R. Borgaonkar. An analysis of the Asprox botnet. In *Fourth International Conference on Emerging Security Information Systems and Technologies (SECURWARE)*, pages 148–153, 2010.

[9] A. Cassandra. POMDPs: Who needs them? www.pomdp.org/pomdp/talks, 2003. Online; accessed 27-November-2012.

[10] A. Cassandra, L. Kaelbling, and M. Littman. Acting optimally in partially observable stochastic domains. In *American Association for Artificial Intelligence*, pages 1023–1028, 1994.

[11] T.M. Chen and V. Venkataramanan. Dempster-Shafer theory for intrusion detection in ad hoc networks. *Internet Computing, IEEE*, 9(6):35–41, 2005.

[12] B. Cheng, G. Liao, C. Huang, and M. Yu. A novel probabilistic matching algorithm for multi-stage attack forecasts. *IEEE Journal on Selected Areas in Communications*, 29(7):1438–1448, 2011.

[13] X. Cheng and B. Yang. Anti-honeypot based on honeypot characteristic. *Modern Electronics Technique*, 15:89–92, 2009.

[14] M. de Vivo, E. Carrasco, G. Isern, and G.O. de Vivo. A review of port scanning techniques. *Computer Communication Review, ACM SIGCOMM*, 29(2):41–48, 1999.

[15] 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.

[16] M. Feily, A. Shahrestani, and S. Ramadass. A survey of botnet and botnet detection. In *3rd International Conference on Emerging Security Information, Systems and Technologies, SECURWARE*, pages 268–273, 2009.

[17] P. Ferrie. Attacks on more virtual machine emulators. *Symantec Advanced Threat Research*, 2006. [Online; accessed 30-January-2012].

[18] J. Franklin, V. Paxson, A. Perrig, and S. Savage. An inquiry into the nature and causes of the wealth of internet miscreants. In *14th ACM conference on Computerand Communications Security*, pages 375–388, 2007.

[19] 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.

[20] O. Hayatle, H. Otrok, and A. Youssef. A game theoretic investigation for high interaction honeypots. In *IEEE International Conference on Communications (ICC), 2012*, pages 6662–6667, 2012.

[21] O. Hayatle, A. Youssef, and H. Otrok. Dempster-Shafer evidence combining for (anti)-honeypot technologies. *Information Security Journal: A Global Perspective*, 21(6):306–316, 2012.

[22] T. Holz and F. Raynal. Detecting honeypots and other suspicious environments. In *Information Assurance Workshop (IAW). Proceedings from the 6th Annual IEEE SMC*, pages 29–36, 2005.

[23] D. Jang, M. Kim, H. Jung, and B. Noh. Analysis of http2p botnet: case study waledac. In *IEEE 9th Malaysia International Conference on Communications (MICC)*, pages 409–412, 2009.

[24] S. Jha, O. Sheyner, and J. Wing. Two formal analysis of attack graphs. In *Computer Security Foundations Wrokshop*, pages 49–63, 2002.

[25] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G.M. Voelker, V. Paxson, and S. Savage. Spamalytics: An empirical analysis of spam marketing conversion. In *15th ACM Conference on Computer and Communications Security*, pages 3–14, 2008.

[26] N. Krawetz. Anti-honeypot technology. In *IEEE Security & Privacy Magazine*, pages 76–79, 2004.

[27] O.P. Kreidl. Analysis of a Markov decision process model for intrusion tolerance. In *Dependable Systems and Networks Workshops (DSN-W)*, pages 156 –161, 2010.

[28] 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.

[29] C. Li, W. Jiang, and X. Zou. Botnet: Survey and case study. In *2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*, pages 1184–1187, 2009.

[30] 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.

[31] Z. Liu, C. Wang, and S. Chen. Correlating multi-step attack and constructing attack scenarios based on attack pattern modeling. In *International Conference on Information Security and Assurance (ISA)*, pages 214–219, 2008.

[32] I. Mokube and M. Adams. Honeypots: Concepts, approaches, and challenges. In *45th Annual Southeast Regional Conference, SIGAPP: ACM Special Interest Group on Applied Computing. ACM*, pages 321–326, 2007.

[33] S. Mukkamala, K. Yendrapalli, R. Basnet, M.K. Shankarapani, and A.H. Sung. Detection of virtual environments and low interaction honeypots. In *Information Assurance and Security Workshop (IAW), Proceedings of the 8th Annual IEEE SMC*, pages 92–98, 2007.

[34] M. J. Osborne and A. Rubistein. *A Course in Game Theory*. MIT Press, 1994.

[35] H. Otrok, B. Zhu, H. Yahyaoui, and P. Bhattacharya. An intrusion detection game theoretical model. *Information Security Journal: A Global Perspective*, 18(5):199–212, 2009.

[36] M. Petrik, G. Taylor, R. Parr, and S. Zilberstein. Feature selection using regularization in approximate linear programs for Markov decision processes. In *27th International Conference on Machine Learning*, pages 871–878, 2010.

[37] R. P'ıbil, V. Lisý, C. Kiekintveld, B. Bošanský, and M. Pěchouček. Game theoretic model of strategic honeypot selection in computer networks. In *Decision and Game Theory for Security*, volume 7638 of *Lecture Notes in Computer Science*, pages 201–220. Springer Berlin Heidelberg, 2012.

[38] N. Provos. A virtual honeypot framework. In *13th Conference on USENIX Security Symposium - Volume 13*, pages 1–14, 2004.

[39] M.L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc., 1994.

[40] N.S. Raghava, D. Sahgal, and S. Chandna. Classification of botnet detection based on botnet architechture. In *International Conference on Communication Systems and Network Technologies (CSNT)*, pages 569–572, 2012.

[41] N.C. Rowe, B.T. Duong, and E.J. Custy. Fake honeypots: A defensive tactic for cyberspace. In *Information Assurance Workshop (IAW), Proceedings of the 7th Annual IEEE SMC*, pages 223–230, 2006.

[42] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[43] T.J Sheskin. *Markov Chains and Decision Processes for Engineers and Managers*. CRC Press, 2011.

[44] G.N. Shirazi, P.Y. Kong, and C.K. Tham. Cooperative retransmissions using Markov decision process with reinforcement learning. In *2009 IEEE 20th International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 652–656, 2009.

[45] L. Spitzner. Honeypots, definitions and value of honeypots. http://www.tracking-hackers.com/papers/honeypots.html. Online; accessed 10-January-2013.

[46] A. Srivastava and J. Giffin. Tamper-resistant, application-aware blocking of malicious network connections. In *Recent Advances in Intrusion Detection*, volume 5230, pages 39–58. Springer Berlin Heidelberg, 2008.

[47] 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.

[48] R. Vogt, J. Aycock, and M. Jacobson. Army of botnets. In *Network and Distributed System Security Symposium*, pages 111–123, 2007.

[49] G. Wagener, R. State, A. Dulaunoy, and T. Engel. Self adaptive high interaction honeypots driven by game theory. In *SSS*, pages 741–755, 2009.

[50] Z. Zhu, G. Lu, Y. Chen, Z.J. Fu, P. Roberts, and K. Han. Botnet research survey. In *32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, pages 967–972, 2008.

[51] C.C. Zou and R. Cunningham. Honeypot-aware advanced botnet construction and maintenance. In *International Conference on Dependable Systems and Networks (DSN)*, pages 199–208, 2006.