Game Theoretic Model of Strategic Honeypot Selection in Computer Networks

Radek Píbil[1], Viliam Lisý[1], Christopher Kiekintveld[2], Branislav Bošanský[1], and Michal Pěchouček[1]

[1] Agent Technology Center, Department of Computer Science and Engineering Faculty of Electrical Engineering, Czech Technical University in Prague Czech Republic
{radek.pibil, viliam.lisy, branislav.bosansky, michal.pechoucek}@agents.fel.cvut.cz

[2] Department of Computer Science, University of Texas at El Paso (UTEP), United States of America
cdkiekintveld@utep.edu

Abstract A honeypot is a decoy computer system used in network security to waste the time and resources of attackers and to analyze their behaviors. While there has been significant research on how to design honeypot systems, less is known about how to use honeypots strategically in network defense. Based on formal deception games, we develop two game-theoretic models that provide insight into how valuable should honeypots look like to maximize the probability that a rational attacker will attack a honeypot. The first model captures a static situation and the second allows attackers to imperfectly probe some of the systems on the network to determine which ones are likely to be real systems (and not honeypots) before launching an attack. We formally analyze the properties of the optimal strategies in the games and provide linear programs for their computation. Finally, we present the optimal solutions for a set of instances of the games and evaluate their quality in comparison to several baselines.

Keywords: honeypots, game theory, network security, deception

1 Introduction

Society increasingly depends on information technology and computer networks to deliver vital information and services. Protecting these systems and the information they contain is a growing priority, even as they become more attractive targets for criminal activity. Cybercriminals are highly motivated and devote large efforts to launching sophisticated attacks, requiring network administrators to adopt increasingly sophisticated countermeasures to protect their networks. Honeypots are one of these countermeasures that provide a unique set of benefits for network defense. Falling costs for deploying honeypots and improved virtualization technologies are likely to lead to increased use of honeypots, including systems with many honeypots on a single network.

A honeypot is a computer system placed on a network explicitly in order to attract the attention of an attacker. It does not store any valuable data and it thoroughly logs everything that happens in the system. Honeypots help to increase the security of computer systems in two ways [1]: (1) The presence of honeypots wastes the attacker's time and resources. The effort an attacker spends to compromise a honeypot and learn that it does not contain any useful information directly takes away time and resources that could be used to compromise valuable servers. (2) Moreover, once the attacker compromises a honeypot, the network administrator can analyze all of the attacker's actions in great detail, and use the information obtained to better protect the network. For example, specific security holes used in an attack can be patched, and new attack signatures added to antivirus and intrusion detection systems. Attacks on honeypots can also serve as an "early warning" system for administrators, providing more time to react to attacks in progress.

For these reasons, the network administrators using honeypots try to maximize the probability that the attacker attacks a honeypot and not a real server. However, with an increasing use of this technology, attackers have started to consider the existence of honeypots during their attacks and take steps to avoid attacking them. For example, once they gain access to a system, they can use multitude of methods to probe the system and rule out the possibility that they are in a honeypot before they continue with their attack (e.g., [2]). To be effective against more sophisticated attackers, honeypots must be sufficiently disguised that they are not obvious (i.e., they cannot simply present the most vulnerable possible target). This leads us to analyze using honeypots from an adversarial perspective, where network administrators reason about the strategies of the attackers and vice versa.

Game theory is a formal framework developed to analyze interactions between multiple decision makers. In this paper, we present two novel game-theoretic models of adding honeypots to a network and the following target selection by the attacker. The first model combines a resource allocation game and a deception game, and is designed to answer basic question about how many honeypots a defender should use, and how they should be configured. In particular, we consider the possibility that honeypots can be configured to look like real targets of varying importance, offering new ways to deceive an attacker. The second model extends the first one to add the capability for an attacker to strategically probe targets before launching an attack to determine whether they are likely to be honeypots or real servers. Both models are formulated as zero-sum extensive-form imperfect-information games, and we provide linear programs for computing the optimal strategies of the players (i.e., the network administrator and the attacker) in both cases.

We solve the linear programs using a state-of-the-art optimization toolkit (*CPLEX*). This provides greater scalability than previous models [3] that were solved using Gambit[4], allowing us to analyze the models in greater detail. These previous models found simple uniform randomization strategies to be optimal for honeypot placement. However, our models show richer and more complex

strategies are necessary when we generalize the assumptions to include non-uniform server values and sophisticated attackers with probing capabilities. Our empirical evaluation shows that the game-theoretic strategies are significantly better in reducing the expected harm of the attacks and they allow using a larger numbers of honeypots more efficiently than two heuristic approaches. We also test our strategies against simple heuristic attackers, in addition to optimal ones. Based on the analysis of the optimal game-theoretic strategies, we provide recommendations to the network administrators applying honeypots in their networks.

The next section explains the relation of the presented research to the previous work. In Section 3, we introduce the basic model without probing, we analyze its properties and present the solution LP. In Section 4, we introduce the possibility of probing. The evaluation and analysis of the optimal strategies for a set of instances of the models is presented in Section 5 and we conclude the paper in Section 6.

2 Related Work

Many software packages for creating honeypots and analyzing attackers' behavior are available through the honeynet project website[3]. This paper does not focus on the technical aspects of creating honeypots, so we do not review this line of research here. An extensive introduction to the practices and technological challenges of applying honeypots is available in [1]. We focus our review on more closely related work that applies game theory to honeypots.

2.1 Honeypots and Game Theory

There are relatively few papers that explore using game theory for creating and deploying honeypots. They can be divided into two categories. One models the interaction within a honeypot during an ongoing attack. The other models the situation before the actual attack, when the attacker selects a target.

In the first category, game theory is used to optimize the information learned about the attacker's strategies by modeling the progress of the attack. In [5] the authors give the defender the option to block the action, or let it be executed, while the attacker can either retry, continue, or stop the attack. In [6] the defender models the attack as a movement on a graph and tries learn the attacker's strategy by making some of the graph nodes more desirable using simulated user activity.

The approach presented in this paper belongs to the second category, in which the game theory is used to optimize the probability that the attacker will attack a honeypot and not a real system. In [3], the authors model situations similar to the ones we model in this paper. However, their model is simpler and results in simple, uniform strategies. They analyze the problem of allocating the real servers and honeypots to the space of IP addresses. However, the attacker cannot

[3] www.honeynet.org

distinguish between individual servers and honeypots, so the only meaningful strategy the attacker can use is to attack a random server. Only if the defender gives the attacker some hint based on the address of the servers, e.g., by assigning the honeypots to the lowest IP addresses, a rational attacker can deviate from a random strategy. Therefore, a rational defender also allocates addresses randomly. In reality, however, not all computers in the network are identical to the attacker. In our model, we consider the importance of the computers, which make the optimal strategies non-trivial and much harder to compute.

In the second part of [3] as well as in [7], the authors give the attacker the option of probing the servers before the attack. The result of a probe is whether the server is real or a honeypot, but the authors assume that the result is fully determined by the defender and not the reality. This implies that the probe results are only useful if the defender voluntarily discloses some information to the attacker. A rational defender uses uniform random probe results and the attacker ignores them. A more realistic assumption is that the defender can successfully deceive the attacker only with certain probability. Otherwise, his probe will identify the real nature of the server. In this paper, we consider this generalization and it results to non-trivial strategies for both players.

2.2 Related Game Theoretic Models

A similar task to the honeypot selection is the deployment of false targets in warfare as studied in [8] (among many others). Targets are identically valued as in [3]. The defender selects the number of fake targets to deploy, for which he has to pay from a resource pool that he also has to use for the protection of genuine targets. The attacker chooses the number of targets to attack. However, the attacker is also limited by his own resource pool, and may possibly be able to attack only a single target as in our case. The paper focuses on a proper resource allocation between protection and defense, not the protection strategy, which is uniform. Our focus is on protection strategies taking values of targets into account.

The game theoretic models presented in this paper are a special case of imperfect-information extensive-form games (EFG) with chance nodes. The state-of-the-art algorithm for solving these games optimally is the mathematical program for sequence-form representation of the games [9]. More efficient algorithms can be found for sub-classes of EFGs with special structure. Two such subclasses are the Bayesian Stackelberg games [10] and signaling games [7]. As in our game models, these games include hidden information available only to one of the players, however, this information modifies only the payoffs of the players and not the applicable actions. In our games, the hidden information defines the applicable actions as well, which makes the techniques developed for Bayesian Stackelberg games inapplicable.

A less studied class of games that are most closely related to our models are deception games. A formal deception game was first formulated as an open problem in [11]. One player is given a vector of three random numbers from uniform distribution on unit interval. It changes one of the numbers to an arbitrary

number from the interval and presents the modified vector to the second player. The second player chooses one position in the vector and receives as its reward the number that was originally on that position. The open question stated in the paper is whether there is a better strategy than randomly choosing one of the positions. This question was answered in [12] and a few similar questions about various modifications of the model were published in the next years, but the results generally apply only to the specific game formulations and they do not present the complete strategies to play the game.

3 Honeypot Selection Game

The *Honeypot Selection Game* models a situation where an attacker is deciding which server in a computer network to attack. However, the administrator has added a set of honeypots to the network, and wants to configure them to maximize the probability that the attacker chooses to attack a honeypot rather than a real computer. There are two basic kinds of honeypots. A *low interaction* honeypot is relatively simple, and therefore it can be added to the network at a low cost [13], but even a simple probing by the attacker will reveal it is not a real system. A *high interaction* honeypot is much more expensive to create and maintain. In order to make it believable, authentic user activity and network traffic has to be simulated. Therefore, high interaction honeypots are a limited resource and it is important to optimize their deployment. We focus on the latter category in this paper.

One of the important features of real-world networks is that they have many different types of servers with different configurations (available services, hardware, etc.). Some categories of servers are more important than others, both to the owner of the network and as targets for the attacker. For example, a database server containing valuable customer information would be of a very high value, while a standard desktop computer acting as a server may have a relatively low value. To model this, we assume that each server in the network can be classified into one of a few categories of *importance*, which can be assigned a numeric value that represents the gain/loss associated with a successful attack. One of the decisions that the defender makes when deploying honeypots on a diverse network is how to disguise the honeypots – in other words, which category of server should each honeypot be designed to look like?

We represent a *configuration* of the network by a vector of values representing the apparent importance of each server. The defender knows the values of each of the real servers in the network, and is able to extend the vector of values by adding honeypots. For each honeypot, the defender is able to select the value of the server that will be observed by the attacker (by configuring the honeypot to emulate servers of that category). We assume that both players have knowledge of the typical configurations of the network, so both players know the distribution of values in the network. For any configuration, the players can calculate the probability that the configuration is the actual configuration of the network. We also assume that the defender uses a fixed number of honeypots to add to the

network, and that the attacker knows the number of honeypots (but not their assigned values). This is a worst case assumption about the attacker, and the model could be generalized to allow for imperfect information about the number of honeypots, though it makes the problem more difficult to solve.

Consider the following example. The network has two servers, which have importance values 4 and 3. The administrator has one honeypot to deploy, and needs to decide how to configure it, which corresponds to assigning a value in our model. He could assign it a value of 5 to make it appear very attractive (e.g., by making it appear that contains valuable data and exposing obvious vulnerabilities). The attacker observes the unordered vector of values by doing a scan of the network, including the value of the honeypot: (5,4,3). A naïve attacker might attack the server with the highest value (5), therefore attacking the honeypot. However, a sophisticated attacker might reason that this is "too good to be true" and choose instead to attack the next best server, with a value of 4. If the attacker chooses a real server to attack, he obtains a reward and the network administrator is penalized. If the attacker chooses to attack a honeypot, he does not obtain any reward and possibly is penalized for disclosing his attack strategy. We model the game as a zero-sum game, so a gain for one player is a loss for the other. While this may not always be the case, it allows for faster solution methods and can provide a solution with guaranteed quality against any (not necessarily rational) opponent. From this example, we can see that the defender's goal is to somehow convince the attacker to selecting a honeypot, and that assigning all honeypots the maximal value may not be the optimal strategy.

3.1 Formal Definition of the Honeypot Selection Game

The Honeypot Selection Game (HSG) is a two-player zero-sum extensive-form game with imperfect and incomplete information.

Definition 1. *The Honeypot Selection Game (HSG) is defined by the tuple* $G = (\boldsymbol{d}, \boldsymbol{a}, \boldsymbol{n}, \boldsymbol{k}, \boldsymbol{D}, p, \mathcal{I}, \chi, \boldsymbol{A}, u)$*:*

- $\mathbf{d}, \mathbf{a}$ *are the players in the game called the defender and the attacker;*
- $\mathbf{n}$ *is the number of real servers;*
- $\mathbf{k}$ *is the number of honeypots;*
- D *is a set of importance values;*
- $p : D^n \to [0, 1]$ *is the probability of each configuration of real servers;*
- $\mathcal{I}$ *is a set of all attacker information sets* ($I \in \mathcal{I}$, $I \subseteq D^{n+k} = D^s$)*;*
- $\chi : D^n \to \mathcal{P}(\mathcal{I})$ *is a function that provides a set of possible actions for the defender, which appends a set of honeypot values to the observed* $\boldsymbol{x} \in D^n$*;*
- $\mathbf{A}$ *is a union of all possible attacker actions for all* $\boldsymbol{y} \in \mathcal{I}$*;*
- $u : D^n \times \mathcal{I} \times \mathbf{A} \to \mathbb{R}^+$ *is the expected utility function for the attacker* ($-u$ *is the utility function for the defender), defined if the second parameter is in* $\chi(\boldsymbol{x})$ *with* $\boldsymbol{x}$ *being first parameter.*

Nature starts by randomly choosing the network configuration $\boldsymbol{x} \in D^n$ according to a known probability distribution p. The defender learns the value $\boldsymbol{x}$

and chooses values for the k honeypots to apply. The defender can insert honeypots anywhere in vector $\boldsymbol{x}$, creating a vector $\boldsymbol{y}$ of length $s = n + k$, which is presented to the attacker. The attacker then chooses one server to attack from $\boldsymbol{y}$. If he attacks a real server i, he obtains the reward y_i from $\boldsymbol{y} = (y_1, \ldots, y_i, \ldots, y_s)$. If he attacks a honeypot the attacker obtains a reward of 0.

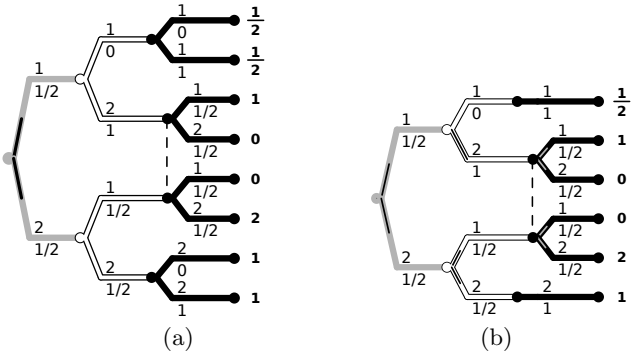


Figure 1. (a) The game tree of a Honeypot Selection Game rendered by Gambit [4] with one real server, one honeypot and a domain $\{1, 2\}$. Light gray edges are random choices, white edges are defender's actions and black edges the attacker's actions. Server values corresponding to actions are above the branches, while probabilities are under them. (b) The same game with grouped attacker's actions.

Extensive form games are usually represented as game trees. An example of a small HSG with one real server ($n = 1$), one honeypot ($k = 1$) and two importance values ($D = \{1, 2\}$) is shown in Figure 1(a). The root node of the game is a chance node with two outcomes representing the nature. In the example, two configurations are possible and the distribution p depicted below the branches is uniform. In each possible real configuration (i.e., child of the root), the defender chooses to add a set of honeypots to the network and defines the information set the attacker will be in (χ). In the example, the defender can add one honeypot with a value 1 or 2. The possible attacker information sets are $\mathcal{I} = \{11, 12, 22\}$.

We assume that the ordering of the vector is arbitrary and contains no information, so (12) and (21) are equivalent. The tree nodes in the same information set are connected by dashed line. In each information set, the attacker can attack one of the servers, which is the set of actions $\mathbf{A}$. In the example, in the top information set (11), the attacker can choose the first server with importance 1 or the second server with importance 1. One of them is real, but they are indistinguishable, so the expected payoffs for either is $u(1, 11, *) = \frac{1}{2}$. In the middle information set, the attacker can choose to attack the server with a value 1 or 2. In the top node of the information set, he can gain 0 or 1, in the bottom, he can gain 0 or 2, but he cannot distinguish between these two nodes, and must use the same strategy in both.

3.2 Solution of the Game

A *strategy* of a player in a game defines what action the player performs in any situation that can occur in the game. A *solution* of a game is a set of strategies, one for each player, that satisfies some notion of optimality. We will search for the solution in form of a *behavioral strategy*[9], i.e., a strategy that prescribes a probability distribution over all possible actions in each possible situation. For the defender in our game, this means determining the probability of using each combination of available honeypot values for each possible configuration of the real part of the network. We allow *mixed strategies* (i.e., randomized strategies), since they generalize pure strategies and allow for strategic deception in adversarial games. The goal of the defender is to maximize his expected payoff, which in a zero-sum game corresponds to minimizing the attacker's expected payoff. A pair of strategies that achieves these maximal/minimal expected payoffs is a Nash equilibrium of the game.

3.3 Properties of the Honeypot Selection Game

We present some useful properties of the HSG game. Our analysis provides intuition about meaningful strategies by identifying sets of dominated actions. Removing these dominated strategies also allows us to reduce the size of the game and improve scalability of solution methods.

Lemma 1. *If the attacker sees a vector of values $\boldsymbol{y} \in D^s$, then he has a strategy that guarantees payoff*

$$m(\boldsymbol{y}) = \max_{S \subseteq \{1,\dots,s\}} U(S, \boldsymbol{y}), \text{ where } U(S, \boldsymbol{y}) = \frac{\sum_{i \in S} y_i - \sum max(k, \boldsymbol{y})}{|S|}. \quad (1)$$

The function $max(k, \boldsymbol{y})$ takes k maximum values from $\boldsymbol{y}$.

Proof. $U(S, \boldsymbol{y})$ is the lower bound on the value of the attacker's strategy that uniformly randomizes over the targets in S, which is met if the most important targets in the set are honeypots. The attacker can choose the best S with the information he has available and obtain $m(\boldsymbol{y})$. □

Lemma 2. *The maximizing S from Lemma 1 does not contain any index of a server with a value lower than $m(\boldsymbol{y})$.*

Proof. If $m(\boldsymbol{y}) \leq 0$ the lemma holds trivially so WLOG $|S| > k$. For contradiction assume the maximizing S contains index j, such that $y_j < m(\boldsymbol{y})$. Then $m(\boldsymbol{y}) = U(S, \boldsymbol{y}) \Rightarrow (|S| - 1)m(\boldsymbol{y}) = (\sum_{i \in S \setminus \{j\}} y_i - \sum max(k, \boldsymbol{y})) + y_j - m(\boldsymbol{y})$ $\Rightarrow m(\boldsymbol{y}) - U(S \setminus \{j\}, \boldsymbol{y}) = (y_j - m(\boldsymbol{y}))/(|S| - 1) < 0$. Hence $m(\boldsymbol{y}) < U(S \setminus \{j\}, \boldsymbol{y})$ which contradicts with S being maximizing. □

Corollary 1. *Attacking a target with a value lower than $m(\boldsymbol{y})$ can never appear with non-zero probability in any attacker's optimal strategies.*

Proof. If any attacker's strategy attacks a server j with $y_j < m(\boldsymbol{y})$ with positive probability then the strategy can be modified to attack the set S from the Lemma 1 with uniform probability anytime is it supposed to attack j. This increases the expected payoff of the strategy, which contradicts its optimality. □

Corollary 2. *If the defender receives a vector $\boldsymbol{x} \in D^n$ of real targets, it does not have to consider honeypots with value lower than $m(\boldsymbol{x})$.*

Proof. If the defender uses honeypots that do not make it to the set S in computing $m(\boldsymbol{x} \cup \boldsymbol{h})$ then $m(\boldsymbol{x} \cup \boldsymbol{h}) = m(\boldsymbol{x})$. If some of them are present in S, $m(\boldsymbol{x} \cup \boldsymbol{h}) \geq m(\boldsymbol{x})$. Either way, the attacker presented by $y = \boldsymbol{x} \cup \boldsymbol{h}$ would not consider attacking a target with value below $m(\boldsymbol{x})$ by Corollary 1. □

Grouping of Server Values We also suggest a more compact representation of the game. Since we assume that the attacker cannot distinguish between the servers of the same value, we reduce the number of the actions available to the attacker in each information set $I \in \mathcal{I}$ to the number of different values in the observed configuration $\boldsymbol{y}$. To do this we create *groups* of servers that have identical importance values (and are therefore indistinguishable). The expected value for choosing any server from that group is computed by assuming that the attacker actually chooses uniformly among members of the group, some of which may be real and some honeypots. Recall the example from Figure 1(a), where the attacker could not distinguish between the real server and the honeypot, both valued 1. We limit the attacker to one action for this information set, with the expected value of $\frac{1}{2}$. The reduced game tree is in Figure 1(b).

3.4 Solution using Linear Programming

We compute a Nash equilibrium of the game in behavioral strategies using a linear program (LP) based on the state-of-the-art method for imperfect-information extensive-form games – a sequence-form LP (e.g., see [9]). The sequence-form utilizes a compact representation of imperfect-information extensive-form games with perfect recall termed *sequences* [14,15], where one sequence for a player represents an ordered list of actions for the player from the root to some node in the game tree. In the following we use the term *compatibility* of sequences – we say that two sequences (one for each player) are compatible, if a step-by-step execution of all the actions in the sequences is a valid course of play. The behavioral strategies can be represented as a probability of executing some sequence conditioned on the opponent playing a compatible sequence. We present two different LP formulations for finding the optimal strategies for the attacker and the defender, assuming in each case that the opponent plays a best response.

Defender's Linear Program The LP for computing the defender's strategy is as follows. There are two types of variables: (1) $v_I \in \mathbb{R}^+$ represents an expected value of a subgame assigned to each information set of the attacker $I \in \mathcal{I}$, and

(2) variables $p_{d_I^{\boldsymbol{x}}} \in [0,1]$ represent the probability of the defender choosing set I (adding a specific honeypots) for each possible real configuration of the network $\boldsymbol{x} \in D^n$. Furthermore, u denotes the utility function of the attacker that defender minimizes, and $\chi^{-1}(I) : \mathcal{I} \mapsto \mathcal{P}(D^n)$ denotes an inverse function that maps an information set to a set of possible configurations of the real part of the network. Finally, $p_{\boldsymbol{x}}$ denotes the probability of network configuration $\boldsymbol{x}$.

$$\min_{v,d} \sum_{I \in \mathcal{I}} v_I \tag{2a}$$

$$v_I \geq \sum_{\boldsymbol{x} \in \chi^{-1}(I)} u(\boldsymbol{x}, I, a_i^I) p_{d_I^{\boldsymbol{x}}} \quad \forall I \in \mathcal{I},\ \forall a_i^I \text{ action applicable in } I \tag{2b}$$

$$\sum_{I \in \chi(\boldsymbol{x})} p_{d_I^{\boldsymbol{x}}} = p_{\boldsymbol{x}} \quad \forall \boldsymbol{x} \in D^n \tag{2c}$$

The program minimizes the utility of the attacker by searching for the optimal strategy of the defender $p_{d_I^{\boldsymbol{x}}}$. These variables are constrained by (2c) in order to represent valid probabilities of sequences played by the defender, conditioned on the other players playing compatible sequences (both nature and the attacker). Finally, the attacker chooses the optimal solution in each information set I. Hence, the expected value v_I is maximized for all possible configurations and actions of the attacker in constraints (2b).

Attacker's Linear Program The LP for computing the optimal strategy for the attacker is similar – the attacker is maximizing its utility value through probabilities for each action $p_{a_i^I} \in [0,1]$ in each information set I, while the defender selects an optimal action minimizing the expected utility value at each information set corresponding to each network configuration $\boldsymbol{x}$ in constraints (3b).

$$\max_{v,a} \sum_{\boldsymbol{x} \in D^n} p_{\boldsymbol{x}} v_{\boldsymbol{x}} \tag{3a}$$

$\forall I \in \mathcal{I}$ assume the attacker can perform actions $\{a_1^I, \ldots, a_m^I\}$:

$$\sum_{i \in \{1,\ldots,m_I\}} u(\boldsymbol{x}, I, a_i^I) p_{a_i^I} \geq v_{\boldsymbol{x}} \quad \forall I \in \mathcal{I},\ \forall \boldsymbol{x} \in \chi^{-1}(I) \tag{3b}$$

$$\sum_{i \in \{1,\ldots,m_I\}} p_{a_i^I} = 1 \tag{3c}$$

Size of the Linear Programs The size is exponential with $|\boldsymbol{y}| = s$ in both constraints and variables. This follows from the upper bound of the number of attacker's information sets, $|\mathcal{I}|$, which is at most equal to $|D|^s$. The exponential size of the programs currently limits the applicability of this approach to large computer networks. In this paper, we focus on the validation of the proposed model and we leave further solution computation optimization to future

research. Moreover, if the instance is too large, good strategies can be computed using approximation algorithms, like CFR, instead of LP. The optimal solutions, however, provide better grounds for our initial analysis.

4 Honeypot Selection Game with Probes

In this section we extend the basic model from the previous section by allowing the attacker to analyze the observed servers to learn, whether they are real servers or honeypots. The main idea of the model with probes is that the attacker, prior to the actual attack, can use *probes* to try to discover the true nature of servers, whether a probed server is real (denoted R), or a honeypot (HP).

We assume that the attacker can use a limited number of probes, and that the results of the probes are stochastic. The first assumption reflects the limited time and resources the attacker typically has for the attack before being exposed. The second assumption models the fact that the attacker cannot be perfectly sure if the server is a honeypot or not, even after gathering some information through probing.

4.1 Formal Definition of Honeypot Selection Game with Probes

The formal definition of Honeypot Selection Game with Probes (HSGp) follows:

Definition 2. *The HSGp is defined by the tuple* $G = (\Gamma, q, \mathcal{I}_\mathcal{E}, \mathbf{A_p}, \mathbf{A_a}, \psi, u)$*:*

- $\Gamma = (\boldsymbol{d}, \boldsymbol{a}, \boldsymbol{n}, \boldsymbol{k}, \boldsymbol{D}, p, \mathcal{I}, \chi, \boldsymbol{A}, u)$ *is a basic HSG;*
- q *is the number of probes to be performed by the attacker;*
- $\mathcal{I}_\mathcal{E}$ *is a set of all attacker information sets,* $\mathcal{I} \subseteq \mathcal{I}_\mathcal{E}$*;*
- $\mathbf{A_p}$ *is a set of all possible attacker probing actions;*
- $\mathbf{A_a}$ *is a set of all possible attacker attacking actions (not the same as* $\mathbf{A}$*, because of probed servers, explained in this section);*
- $\psi : \{R, HP\}^i \times {\mathbf{A_p}}^{i+1} \to [0,1]; \forall i \in \{0, \ldots, q-1\}$ *is a function that assigns the probability of a probe result being* R*, based on the history of probing decisions and observations;*
- $u' : D^n \times \mathcal{I} \times ({\mathbf{A_p}}^q, \mathbf{A_a}) \times \{R, HP\}^q \to \mathbb{R}^+$ *is the expected utility function for the attacker (*$-u$ *for the defender). The observations are necessary, because the servers of the same value are indistinguishable as explained in this section.*

In order to define ψ, we assume that results of probing a single fixed server are independent and identically distributed to simplify the mathematical expression (though in principle the model is not restricted to this). The probability that a probe to a server that is either R or HP returns either result R or HP is fixed and does not change with repeated attempts. We denote these probabilities $\alpha(R|R)$ – the probability of R when probing a real server – and $\alpha(HP|HP)$ – the probability of HP when probing a honeypot. The complementary probabilities for false positives and false negatives (misidentification) follow from these.

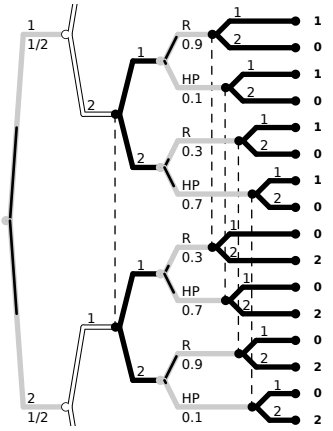


Figure 2. One root information set with observed values 1 and 2 for the attacker including subtrees for the node from the set. R and HP are the outcomes of probes.

Figure 2 shows part of a game tree for an instance of honeypot selection game with probes. The attacker chooses a server to probe in its information set (12), followed by chance nodes representing the uncertain results of the probes. The probability values for the chance nodes that determine the results of the probes are given according to the function ψ. Although we assume that the probe results from a fixed server are independent from each other and they are determined by the parameters α, the probabilities in the game tree depend on the path in the tree that lead to them. In the following section we describe a methodology for computing ψ based on α and observations.

4.2 Probabilities of the Chance Nodes After Probing Outcomes

The probing model also handles a set of servers of a single value as indistinguishable. Probing a server distinguishes it from, but the rest remains indistinguishable. Each of these servers has a probability of being real, at first depending only on the number of honeypots among them. The probing modifies these probabilities and directly affects ψ. Let us describe the methodology for defining the ψ function more formally. We focus on a single set of servers sharing the same importance value ϕ. We base our notation on previous definitions: k_ϕ is the number of honeypots, n_ϕ is the number of real servers, $s_\phi = k_\phi + n_\phi$ is the total.

The prior probability of the i-th server with value ϕ being real is $p(i)$. The ordering is drawn at random uniformly to make sure that it cannot be exploited. We denote $p(i|\boldsymbol{o}, \boldsymbol{b})$ as the posterior probability of the i-th server being real after a sequence of observations $\boldsymbol{o} = (o_1, \ldots, o_l)$ and probing actions $\boldsymbol{b} = (b_1, \ldots, b_l)$ with l as the l-th probe. The ψ function value, the probability of an outcome, for the first probe of the attacker (examining server i) can be calculated as $\psi(\emptyset, i) = p(R) = p(R|i)p(i) + p(R|\neg i)p(\neg i)$.

Based on the outcome we can update the probabilities $p(i)$ for servers in ϕ. We can use the Bayes rule to calculate $p(i|R)$ for the probed server. For any other server $j \neq i$, the probability of being real after the first probing can be calculated as $p(j|o_1, b_1) = p(j|i)p(i|o_1, b_1) + p(j|\neg i)p(\neg i|o_1, b_1)$; where $p(j|i)$ represents the probability of server j being real if server i is real without any observations calculated as $p(j|i) = \frac{n_\phi - 1}{s_\phi - 1}$, and $p(j|\neg i)$ representing the case, where i is not R. However, this rule becomes difficult to express concisely, with the increasing amount of probes, because calculating $p(j|i, o_{l+1}\boldsymbol{o}, b_{l+1}\boldsymbol{b})$ becomes very difficult.

To see why this is the case, let us denote each of the possible assignments of real servers and honeypots for ϕ by *characteristic vectors* $c \in \{R, HP\}^{s_\phi}$. Let us put each of the vectors into groups that have honeypots and real servers in the same places for all probed locations. For example, the first server was probed and yields two groups of characteristic vectors, one with a honeypot as the first server, and one with a real server as the first server. Each newly probed server subdivides the groups further. Each subdivided group requires a separate Bayesian update. There will be at most 2^{s_ϕ} groups, each representing a single characteristic vector per group.

To exactly calculate all the probabilities $p(i|o_{l+1}\boldsymbol{o}, b_{l+1}\boldsymbol{b})$ after $(l+1)$-th probe, we consider all characteristic vectors that are compatible with the current information set in the game tree. Each game situation has a list of probabilities of being true assigned to each of the characteristic vectors for each of the importance values. The probability $p(i|\boldsymbol{o}, \boldsymbol{b})$ can be calculated by summing over probabilities of characteristic vectors with a real server at the i-th position:

$$p(i|\boldsymbol{o}, \boldsymbol{b}) = \sum_{c \in S} p(c|\boldsymbol{o}, \boldsymbol{b}), \quad S = \{c | \forall c \in \{R, HP\}^{s_\phi}; c_i = R\} \tag{4}$$

With the *i.i.d.* assumption, the updates are based on Bayes' Rule. Vector c is the characteristic vector, whose probability is being updated after probing b_{l+1}.

$$p(c|o_{l+1}\boldsymbol{o}, b_{l+1}\boldsymbol{b}) = \begin{cases} \frac{\alpha(o_{l+1}|R)p(c|\boldsymbol{o},\boldsymbol{b})}{p(o_{l+1}\boldsymbol{o}|b_{l+1}\boldsymbol{b},\boldsymbol{o})}, & \text{iff } c_{b_{l+1}} = R \\ \\ \frac{\alpha(o_{l+1}|HP)(1-p(c|\boldsymbol{o},\boldsymbol{b}))}{p(o_{l+1}\boldsymbol{o}|b_{l+1}\boldsymbol{b},\boldsymbol{o})}, & \text{iff } c_{b_{l+1}} = HP \end{cases} \tag{5}$$

The updated vector of probabilities is used in the subtree of the node.

Grouping with Probes We can reduce the number of actions for the attacker by grouping all servers of the same importance value that have not been probed yet. These are treated identically as the *"next server to be probed"*. They have the same outcomes and same probabilities of being real, so we do not break the interpretation of the game. Every time a new server is probed, it is differentiated from the rest of the servers in the group. This approach keeps a fixed ordering, which the defender still cannot influence.

Properties of HSGp There is an opportunity for further pruning in the HSGp besides creating groups. In the final decision node of the attacker, we can replace

a set of attacks on the servers of a same importance with a single attack that represents an attack on the server with the largest probability of being real. Among all the servers of the same importance value, the one with the highest probability being real (in the node) has the highest expected utility and the observations from probes would also lead the attacker to see it as such; hence, this strategy is dominant and will be selected by a rational attacker.

4.3 Solution Using Linear Programming

The linear program calculating the solution is an extension of the linear program presented in Section 3.4. The extension treats the chance nodes as defender's choice nodes with a fixed strategy. However, it is still necessary to provide constraints for the weighted values for the attacker's choice nodes for probes.

In order to improve readability we denote $fin(\mathcal{I}_\mathcal{E})$ to be a set of all information sets where the attacker chooses the server to attack. $\Sigma_{a,c,I}$ refers to compatible sequences of attacker's actions and chance node outcomes for I, one of the starting information sets for the attacker. Function $orgn(I)$ returns the first information set of the attacker encountered on the path in the game tree to I. $Ext_a(\sigma_a)$ returns the shortest extension to sequence $\sigma_a \in \Sigma_a$, where Σ_a is the set of all possible sequences of attacker's actions. By the shortest extension we mean sequence σ_a with a single, valid attacker action appended to its end. In the program, we also use $\mathcal{I}_\mathcal{E}(\sigma_a)$ as a function that returns a set of information sets reached by the attacker after executing sequence of actions σ_a.

$$\min_{v,d} \sum_{I \in \mathcal{I}} v_I \tag{6a}$$

$$v_I \geq \sum_{\boldsymbol{x} \in \chi^{-1}(orgn(I))} -u'(\boldsymbol{x}, I, \sigma_a, \sigma_e) p_{d_I^{\boldsymbol{x}}} \quad \forall I \in fin(\mathcal{I}_\mathcal{E}), \forall (\sigma_a, \sigma_e) \in \Sigma_{a,e,I} \tag{6b}$$

$$v_{I(\sigma_a)} \geq \sum_{I' \in \mathcal{I}(Ext_a(\sigma_a))} v_{I'} \quad \forall \sigma_a \in \Sigma_a \tag{6c}$$

$$\sum_{I \in \chi(\boldsymbol{x})} p_{d_I^{\boldsymbol{x}}} = p_{\boldsymbol{x}} \quad \forall \boldsymbol{x} \in D^n \tag{6d}$$

The defender aims to minimize the expected utilities of the attacker's best response. We define u' as $u'(\boldsymbol{x}, I, \sigma_a, \sigma_e) = \phi p_e(\sigma_e) p_t(\phi_i)$, where $p_t(\phi_i)$ is the probability of the i-th server in the ϕ-valued set being real in the final decision node t, while $p_e(\sigma_e)$ is the probability of the outcomes of the observations that led to the final information set.

Inequality (6b) provides constraints that maximize the attacker's expected utility in the level just above the one with terminal nodes. The second inequality (6c) provides constraints that maximize over the expected value of the subtrees of attacker's probing decisions by summing over the expected value of each possible probing. The final inequality (6d) makes sure that the probabilities of defender's actions form a valid probabilistic distribution.

Due to space limits, we omit the attacker's LP. The difference between the HSG program and HSGp is in the addition of new constraints that make sure that the probabilities of attacker's sequence are valid in each node, including the chance nodes. The "variables" for chance node probabilities are fixed in each of the chance nodes for each probing outcome. Due to the sequence of q decisions of the attacker, the size of the linear program is exponential in q (and also in s as is the basic HSG).

5 Evaluation of Computed Strategies

In this section we provide experiments and analysis of the behavior of the models with varying parameters. The goal is to identify key characteristics of the game, and compare the quality of the game-theoretic solution to baseline strategies. Finally, we want to derive general principles from the results in order to give the network administrators some rules of thumb for placing the honeypots in a computer network. All of the results are computed using the LP formulations described earlier with *CPLEX 12.1*.

5.1 Experimental Settings

In our experiments we fix the number of real servers, $n = 5$, and the importance values $D = \{1, \ldots, 4\}$. The number of honeypots is $k \in \{0, \ldots, 5\}$. The size of these games is plausible for a small computer network. We use two different probability distributions over the possible network configurations $\boldsymbol{x}$, a uniform distribution and a power-law Yule-Simon distribution with parameter $\rho = 1$. The Yule-Simon distribution reflects a common situation in computer networks with relatively few high-valued targets, and a larger number of less significant targets. For our domain with four values the probabilities from this distribution are in increasing importance order: $(0.6250, 0.2083, 0.1042, 0.0625)$.

As baselines for comparison with the game-theoretic strategies we introduce two methods for each player: (1) *Random* strategy, in which the player always selects a uniform random action in each information set, and (2) *Maximum* strategy, which uses a greedy heuristic always attacking/adding targets with the maximal observed value[4].

Our first set of results compares the payoffs of our baseline strategies and the game-theoretic strategies from two different perspectives. First, we evaluate the guaranteed utility of the different defender's strategies, and then we compare the quality of the attacker's strategies against the defender's Nash equilibrium (*NE*) strategy. The guaranteed utility of a strategy σ is the payoff for the strategy when the opponent plays a best response to σ. We calculate the guaranteed utility using the linear program for the game-theoretic solution, but with fixed probabilities for the defender's actions (and vice versa for the attacker).

[4] Maximal expected value in the case of HSGp, with the same greedy rule for probing.

Our second set of results shows the details of the optimal defender's strategies, i.e., the probability that a honeypot with a specific value will be actually deployed in a computer network. We present the results about honeypot likelihood in two different ways: (1) the probability that *at least one* honeypot of the given value is used by the defender, and (2) the portion of honeypots assigned to each value. In the first case we marginalize over the probabilities of defender's actions that add a honeypot of this value, weighted by the network configuration probabilities. For the second case we marginalize over all defender's actions and weight each component by the number of honeypots of this value added by the action, as well as by the network configuration probabilities. We then renormalize the probabilities (divide by k) so the proportions sum to 1. For example, if an action $d_I^{\boldsymbol{x}}$ adds three honeypots of value 4, its component in the sum is $\frac{3p(d_I^{\boldsymbol{x}})}{k}$.

5.2 Basic HSG

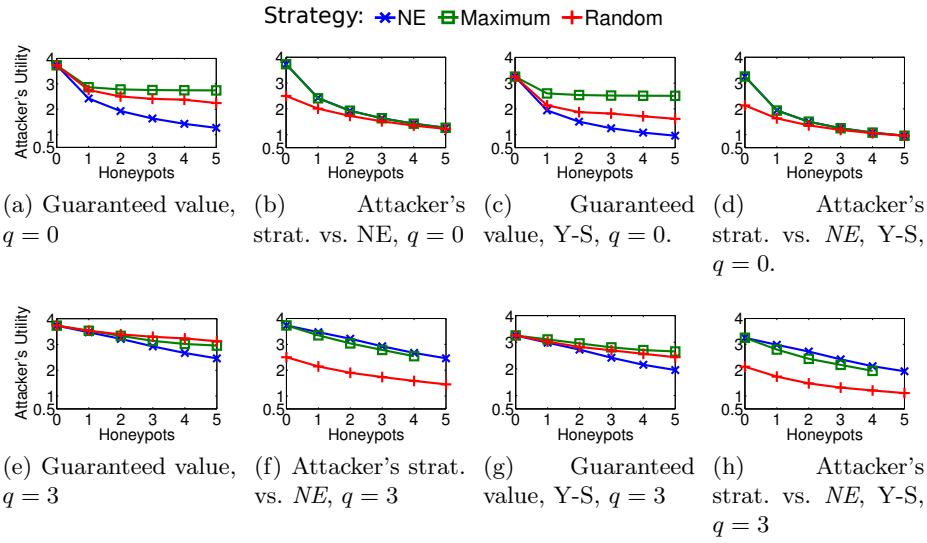
Game Values The results for the game values are presented in Figure 3, first row. In Figure 3(a) we show the guaranteed utility of the defender's *NE*, *Random*, and *Maximum* strategies. The results show that the *Maximum* strategy gains almost no benefit from more than one honeypot, while the *Random* strategy shows a small gain. The *NE* is clearly stronger than the two baselines and significantly increases the defender's utility as the number of honeypots increases.

Figure 3(b) shows the quality of attacker's strategies against the defender's *NE* strategy (attacker prefers higher values). The *Random* strategy performs better with more honeypots, and almost matches the other two strategies when $n = k$. This suggests that the defender's strategy is effectively making it impossible for the attacker to distinguish between the servers based on value. From $k = 0$, *Maximum* strategy has exactly the same payoff as the *NE* strategy, implying that it is part of the support set that the *NE* strategy randomizes over.

The second pair of subfigures (3(c), (d)) shows the game values for the Yule-Simon distribution. For both the guaranteed utility of defender's strategies and the payoffs of the attacker's strategies against the *NE*, the progression is nearly identical with an increasing number of honeypots. The values are smaller overall, which reflects the lower frequency of high-valued servers. The only exception is the *Random* strategy, which improves slightly more than the other strategies, though the *NE* strategy is still better. The overall similarity of results indicates that the choice of distribution does not have a strong effect on the results.

Defender's Strategy Analysis The plots in Figure 4, first row, show how the defender chooses to assign values $D \in \{1, \ldots, 4\}$ to the honeypots. Each line represents one of the four possible values.

Figure 4(a) shows the probability that *at least* one honeypot of the given value is used by the *NE* strategy. We see that it is very rare to use any honeypot 1, as there is little gain from protecting these servers. With few honeypots this is also the case for value 2, but with increased number of honeypots the number 2s becomes more significant. In Figure 4(b), we present the expected proportion of



(a) Guaranteed value, $q = 0$ (b) Attacker's strat. vs. NE, $q = 0$ (c) Guaranteed value, Y-S, $q = 0$. (d) Attacker's strat. vs. *NE*, Y-S, $q = 0$.

(e) Guaranteed value, $q = 3$ (f) Attacker's strat. vs. *NE*, $q = 3$ (g) Guaranteed value, Y-S, $q = 3$ (h) Attacker's strat. vs. *NE*, Y-S, $q = 3$

Figure 3. For all figures $n = 5$, $h = \{1, \ldots, 5\}$, $D = \{1, \ldots, 4\}$. For (a), (b), (c), (d) $q = 0$ (HSG). (a) Guaranteed values of defender's strategies. (b) Values of the attacker's strat. against Nash strategy *NE*. (c) Game values under the Yule-Simon distribution of the R server values with $\rho = 1$. (d) Values of the attacker's strat. vs. *NE* under Y-S. – For (e), (f), (g), (h) $q = 3$ (HSGp). (e) Guaranteed values of defender's strategies. (f) Values of the attacker's strat. vs. *NE*. (g) Game values for the under Y-S. (h) Values of the attacker's strat. vs. *NE* under Y-S.

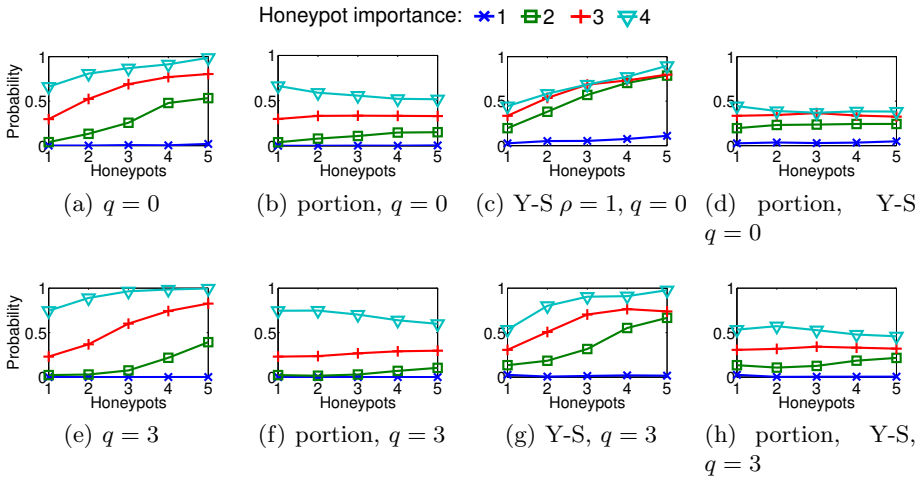
honeypots in the network that have each value. The proportions tend to slightly converge as the number increases, with probabilities of lower valued honeypots increasing, while the probabilities of the higher values decrease. The stability is interesting, as it suggests that network administrators can use the same basic selection ratio over a range of possible amounts of honeypots.

In Figures 4(c) and (d) we show the results under Yule-Simon distribution. The increase in probability of using at least one lower-valued server, primarily at the expense of value 4, is caused by the higher probability they have and therefore the defender protects them more. A slight convergence can be seen here as well, but the portions show less difference than in Figure 4(b).

5.3 HSG with Probes

Next, we present results for the model with probes HSGp. Our set of parameters is $\alpha(R|R) = 0.9$, $\alpha(HP|HP) = 0.7$, with $q = 3$. The probability $\alpha(R|R)$ models the assumption that it is unlikely for an R server to behave as an *HP*. The $\alpha(HP|HP)$ assumes that it is slightly less likely to correctly identify an *HP*.

The attacker's *Random* strategy uniformly probes the servers and then uniformly chooses a server to attack. The attacker's *Maximum* strategy assumes a uniform prior probability of any server being real, regardless of the defender's



(a) $q = 0$ (b) portion, $q = 0$ (c) Y-S $\rho = 1$, $q = 0$ (d) portion, Y-S $q = 0$

(e) $q = 3$ (f) portion, $q = 3$ (g) Y-S, $q = 3$ (h) portion, Y-S, $q = 3$

Figure 4. For all figures $n = 5$, $h = \{1, \ldots, 5\}$, $D = \{1, \ldots, 4\}$. For (a), (b), (c), (d) $q = 0$ (HSG). (a) Probability of use of *HP* values under uniform distribution of R values. (b) The expected portion *HP* values, uniform. (c) Probability of use of *HP* values under Yule-Simon distribution of the R values with $\rho = 1$. (d) The expected portion of *HP* values, Y-S. – For (e), (f), (g), (h) $q = 3$ (HSGp). (e) Probability of use of *HP* values,, uniform. (f) The expected portion of *HP* values, uniform. (g) Probability of use of *HP* values, Y-S. (h) The expected portion of *HP* values, Y-S.

strategy. The strategy probes according to the current highest expected value, conditioned by the observations and probings. After all probes have been used, the server with the maximal expected value is attacked. While this strategy is strategically simple, it has high memory requirements for evaluation because it needs to keep a separate probability vector for each possible plan. The missing data point for $k = 5$ honeypots in Figures 3(f) and (h) is the result.

Game Values The results for the game values are presented in Figure 3, second row. There is an almost linear decrease in attacker's utility in Figure 3(e), which contrasts with the results for $q = 0$, especially for the *NE* strategy (see Section 5.2). The almost-linearity is present also in the attacker's strategies in Figure 3(f). The *Maximum* strategy compares reasonably well with the *NE* strategy for the attacker. The *Random* strategy performs much worse than the other two. These two observations support the use of *NE* attacker's strategy.

Results under both distributions (Figures 3(e), (f) and 3(g), (h)) are very similar. The only difference, apart from the shift towards 0 for the same reason as in HSG, is that *Random* and *Maximum* strategies have exchanged places in Figures 3(e) and 3(g). With $q = 0$, the *Random* strategy is better than *Maximum*, not with $q = 3$. Intuitively higher values need to be protected more, because a probe result gives R the attacker a high confidence that the server is real.

Defender's Strategy Analysis Most of the observations for $q = 0$ hold for $q = 3$ as well. One exception is that the leveling out of value 3 in Figure 4(b) is not present in Figure 4(f). Comparing figures from the first row of Figure 4 ($q = 0$) with the second row ($q = 3$), we can see that with the increased amount of probes, the highest valued 4 is more preferred. We speculate that the reason for this might be the increased chance of the attacker of discerning honeypots from R servers. The selected values for $\alpha(\bullet|\bullet)$ give high probability of a server being observed as R, if it is R ($\alpha(R|R)$), while a slightly lower probability for a HP observed as a HP ($\alpha(HP|HP)$). This could explain why probabilities for 3s do not level out (Figure 4(f)), as opposed to the $q = 0$ case (Figure 4(b)).

6 Conclusion

We introduce new game-theoretic models for analyzing honeypot configuration problems in network security. These models significantly extend previous work in this area, and provide new insights into non-trivial strategies for using honeypots effectively in network security. Our model shows that honeypots should not always be configured to look like the most or least valuable servers in a network, but instead the optimal strategy is randomized and distributes honeypots that look like different types of servers on the network. This becomes increasingly important as networks move towards using a larger number of honeypots as ways to deceive and attract the attention of attackers. This is shown in our empirical results as we see that the Nash equilibrium strategies have a stronger performance relative to baselines as the number of available honeypots increases.

The first model we present is a type of deception game, where the defender tries to disguise honeypots in a network so that the attacker will choose to attack honeypots instead of real servers. Our second model extends this by including probing actions for the attackers, who can try to distinguish honeypots from real servers before actually launching an attack. The probes are noisy, so the attacker still needs to act with imperfect information in these models. We present linear programming models for solving both of these classes of games.

We study the behavior of both of our models empirically, using heuristic baseline strategies for both players. We also vary the assumption about the distribution of importance values on the network. The Nash equilibrium strategies in our models significantly outperform the baseline strategies, regardless of the distribution of values of real servers in the network. We also studied the structure of the equilibrium strategies in these games, which shows that honeypot values in both cases should be distributed across the space of possible configurations. As the number of honeypots increases, there is a change in the strategies, with the optimal strategies placing greater weight on lower values.

Our analysis shows that there are important strategic issues that must be investigated to maximize the efficiency of honeypots in network security, particularly as the purpose of honeypots evolves from learning about attackers to actively deceiving and delaying attackers. It is not sufficient to consider only the

technical issues involved in honeypot design, but also the strategic issues about how they should be used.

Acknowledgments. This research was supported by the Office of Naval Research Global (grant no. N62909-12-1-7019), and the Czech Science Foundation (grant no. P202/12/2054).

References

1. Spitzner, L.: Honeypots: Tracking Hackers. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)
2. Dornseif, M., Holz, T., Klein, C.: NoSEBrEaK - attacking honeynets. In: Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC. (june 2004) 123 – 129
3. Garg, N., Grosu, D.: Deception in Honeynets: A Game-Theoretic Analysis. In: IEEE Information Assurance Workshop. (2007) 107–113
4. McKelvey, R.D., McLennan, A.M., Turocy, T.L.: Gambit: Software Tools for Game Theory. Technical report, Version 0.2010.09.01 (2010)
5. Wagener, G., State, R., Dulaunoy, A.: Self Adaptive High Interaction Honeypots Driven by Game Theory. Stabilization, Safety, and (2009) 1–15
6. Williamson, S.A., Varakantham, P., Hui, O.C., Gao, D.: Active Malware Analysis Using Stochastic Games. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1. AAMAS '12, Richland, SC, International Foundation for Autonomous Agents and Multiagent Systems (2012) 29–36
7. Carroll, T.E., Grosu, D.: A game theoretic investigation of deception in network security. Security and Communication Networks **4**(10) (2011) 1162–1172
8. Hausken, K., Levitin, G.: Protection vs. false targets in series systems. Reliability Engineering & System Safety **94**(5) (2009) 973–981
9. Shoham, Y., Leyton-Brown, K. In: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press (2009) 130–144
10. Paruchuri, P., Pearce, J., Marecki, J., Tambe, M., Ordonez, F., Kraus, S.: Playing games for security: an efficient exact algorithm for solving Bayesian Stackelberg games. In: Proceedings of AAMAS. (2008) 895–902
11. Spencer, J.: A deception game. American Mathematical Monthly (1973) 416–417
12. Lee, K.: On a deception game with three boxes. Int. Jour. of Game Theory **22**(2) (1993) 89–95
13. Cohen, F.: A Mathematical Structure of Simple Defensive Network Deception. Computers & Security **19**(6) (2000) 520–528
14. von Stengel, B.: Efficient Computation of Behavior Strategies. Games and Economic Behavior **14**(2) (1996) 220–246
15. Koller, D., Megiddo, N., von Stengel, B.: Efficient Computation of Equilibria for Extensive Two-Person Games. Games and Economic Behavior **14**(2) (1996) 247–259