Approximate Solutions for Attack Graph Games with Imperfect Information

Karel Durkota[1], Viliam Lisý[1], Branislav Bošanský[2], Christopher Kiekintveld[3]
[1]Agent Technology Center, Dept. of Computer Science, Czech Technical University in Prague
{karel.durkota,viliam.lisy}@agents.fel.cvut.cz
[2]Department of Computing Science, University of Alberta
[3]Department of Computer Science, Aarhus University
bosansky@cs.au.dk
[4]Computer Science Department, University of Texas at El Paso
cdkiekintveld@utep.edu

Abstract

We focus on network hardening with honeypots which could trap and expose the attacker. We model this problem as a general-sum extensive-form game with imperfect information. Based on a specific computer network, the defender seeks for an optimal deployment of honeypots while the attacker uses an attack plan from a library of possible attacks compactly modeled by attack graphs. We detecting the attacker's rationalizable strategies and compute Stackelberg Equilibrium by solving mixed-integer linear programming formulations and improve it with iterative oracle approach. However, this approach has limited scalability. Therefore, we also analyze a trade-off between complexity and optimality of game approximations, which bring useful results for the games in security domain.

1 Introduction

Networked computer systems support a wide range of critical functions in both civilian and military domains. Securing this infrastructure is extremely costly and there is a need for new automated decision support systems that aid human network administrators to detect and prevent the attacks.

We focus on network security hardening problems in which a network administrator (defender) reduces the risk of attacks on the network by setting up honeypots (HPs) (fake hosts or services) in their network [Qassrawi and Hongli, 2010]. Legitimate users do not interact with HPs; hence, the HPs act as decoys and distract attackers from the real hosts. HPs can also send intrusion detection alarms to the administrator, and/or gather detailed information the attacker's activity [Provos, 2004; Grimes *et al.*, 2005]. Believable HPs, however, are costly to set up and maintain. On the other hand, a well-informed attacker anticipates the use of HPs and tries to avoid them. Therefore the problem of deciding which services to duplicate (i.e., *allocating honeypots*) can be modeled using the game-theoretic framework.

Our game-theoretic model extends the long existing line of Stackelberg models used in the physical security domains [Tambe, 2011]. The model presented in this paper advances the state of the art by a novel combination of two elements: (1) we adopt a compact representation of strategies for attacking computer networks called *attack graphs*, (2) the defender uses *deception* by exploiting *imperfect information* the attacker has about the attacked network. While some form of a compact representation have previously been used [Letchford and Vorobeychik, 2013; Durkota *et al.*, 2015], they were assuming the attacker has much more detailed information and almost perfect information about the network. Deception that exploits imperfect information of the attacker about the network have been proposed before [Píbil *et al.*, 2012]; however, the existing model uses a zero-sum assumption and very abstract one step attack actions. Both of these restrictions are removed in our model.

Attack graphs (AGs) compactly represent a rich space of sequential attacker actions for compromising a specific computer network. AGs can be automatically generated based on known vulnerability databases [Ingols *et al.*, 2006; Ou *et al.*, 2006] and they are widely used in the network security to identify the minimal subset of vulnerabilities/sensors to be fixed/placed to prevent all known attacks [Sheyner *et al.*, 2002; Noel and Jajodia, 2008], or to calculate security risk measures (e.g., the probability of a successful attack) [Noel *et al.*, 2010; Homer *et al.*, 2013]. We use AGs as a compact representation of an attack plan library, from which the rational attacker chooses the optimal contingency plan to follow.

The action of the defender is to select which services or hosts will be allocated as HPs in order to minimize the costs for the deployed HPs and successful attacks. We assume that the attacker knows the overall number of HPs, but does not know which types of services defender actually allocated as HPs. This is in contrast to [Durkota *et al.*, 2015], where authors assumed that the attacker knows which types of services contain HPs and the uncertainty was only about which specific server/computer is real or not.

Solving games with stochastic events and imperfect information is generally NP-hard [Letchford and Conitzer, 2010]. Algorithms that compute the optimal solution in this class of games do not scale to real-world settings [Bošanský and Čermák, 2015]. Therefore (1) we present a novel collection of approximations for Stackelberg deception games that are solvable in polynomial time, and (2) we experimentally show that the strategies computed in the approximated models are often very close to the optimal strategies in the original model, e.g., strategy for zero-sum approximation had only

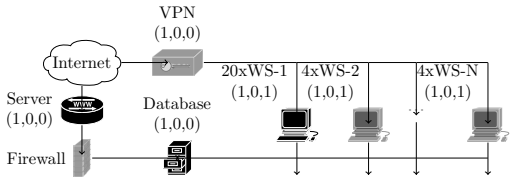


Figure 1: Simple business-like network topology.

0.13% of relative regret. Finally, (3) we propose novel algorithms to compute upper and lower bounds on the expected utility of the defender in the original game to allow the evaluation of the strategies from the approximate models.

2 Background and Definitions

Computer Network

We define network over a set of host types T, such as firewalls, workstations, etc. Two hosts are of the same type if they run the same services and have the same connectivity in the network (i.e., a collection of identical workstations is modeled as a single type). All hosts of the same type present the same attack opportunities, so they can be represented only once in an attack graph. Formally, a computer network $x \in \mathbb{N}_0^T$ contains x_t hosts of type $t \in T$. An exemplar network instance is depicted in Fig. 1, where, e.g., host type WS-1 represents 20 workstations of the same type ($x_{WS-1} = 20$).

Attack Graph

There are multiple representations of attack graphs common in the literature. The *dependency attack graphs* are more compact and allow more efficient analysis than the alternatives [Obes *et al.*, 2013]. Formally, it is a directed AND/OR graph consisting of fact nodes and action nodes. Facts nodes represent logical statement about the network, while actions can make the statements true with corresponding reward to the attacker. Every action a has an associated probability of being performed successfully $p_a \in [0, 1]$, cost $c_a \in \mathbb{R}^+$ that the attacker pays regardless whether the action is successful or not, and a set of host types $\tau_a \subseteq T$ that action a interacts with. The first time the attacker interacts with a type t, a specific host of that type is selected with uniform probability. Future actions with the same host type interact with the same host. There is no reason to interact with a different host of the same type because (1) rewards are defined based on the types, so there is no additional benefit, and (2) interacting with another host increases the probability of interacting with a honeypot and ending the game. The attacker can terminate his attack any time (denoted as action a_T).

We use the common monotonicity assumption [Ammann *et al.*, 2002; Ou *et al.*, 2006; Noel and Jajodia, 2004] that once a fact becomes true during an attack, it can never become false again as an effect of any action.

Attack graphs can be automatically generated by various tools. We use the MulVAL [Ou *et al.*, 2005], which constructs dependency attack graphs from information automatically collected by network scanning tools, such as Nessus[1] or OpenVAS[2]. Previous works (e.g., [Sawilla and Ou, 2008]) show that probabilistic metrics can be extracted from the Common Vulnerability Scoring System [Mell *et al.*, 2006] for different actions available in the National Vulnerability Database [Bacic *et al.*, 2006], historical data, red team exercises, or be directly specified by the network administrator.

Attack policy In order to fully characterize the attacker's attack, we compute a full *contingent attack policy*, which defines an action for each situation that may arise during an attack. This allows identifying not only the actions likely to be executed by a rational attacker, but also the *order* of their execution. Linear plans that may be provided by classical planners (e.g., [Obes *et al.*, 2013; Boddy *et al.*, 2005]) are not sufficient as they cannot represent attacker's behavior after action failures. We define set of all contingent attack policies S_s for network z based on attack graph for the network. *Optimal attack policy* is the attack policy with maximal expected reward for the attacker.

3 Imperfect-Information HP Allocation Game

The goal of the paper is to model an attacker who compromise a computer network for a specific organization (e.g., a company, a bank, a government, etc.). We assume that the set of possible networks in a specific organization and the probabilities of employing each network is a common knowledge to both players. To capture the set of networks in the model, we begin our game with the *nature* choosing a network from the set of networks with the probability corresponding to the probability that the organization employs such network. The defender observes the network and hardens it by adding HPs in it. The attacker observes the network resulting from the choices of nature and the defender, and attacks it optimally based on the attack graph for the observed network.

We model the defender's *optimal HPs allocation problem* as a three-stage game and explain each stage on a game-tree example from Fig. 2.

3.1 Nature Actions

The network basis $b \in \mathbb{N}_0^T$ defines the number of basic host types that every target organization network contains (i.e. it must contain a server, a workstation and a database). In the first stage of the game, nature extends the network basis b by adding $\sum_{t \in T} b_t - n$ hosts, which results in a set of networks X. Each network $x \in X$ consists of n hosts and it is chosen with probability δ_x, which corresponds to how likely network x is employed in a real-world scenario for specific organization.

In Fig. 2 the set of types $T = \{D, W, S\}$ (e.g., database, workstation, server), and the network basis is $b = (1, 0, 1)$, a database and a server. Nature extends network b by adding one host and creates three networks $X = \{(2, 0, 1), (1, 1, 1), (1, 0, 2)\}$, each with uniform probability $\delta_x = \frac{1}{3}$. In the next stage, the defender hardens the network.

3.2 Defender

The defender further extends network $x \in X$ by adding k HPs into it. We define Y to be a set of honeypot networks (networks containing only HPs), where $y \in Y : \sum_{t \in T} y_t = k$. Each HP network $y \in Y$ represents an action that the defender can perform. Thus, he extends each network $x \in X$ with HP network $y \in Y$, which results in a new fused network $z = (x, y)$,

[1] http://www.nessus.org [2] http://www.openvas.org

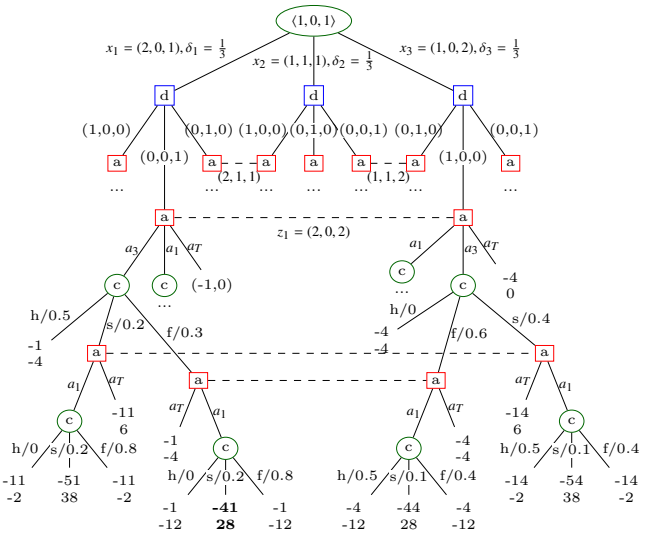


Figure 2: Simple game-tree with host types $T = \{S, D, W\}$, basis $b = (1, 0, 1)$, $n = 3$ and $k = 1$ HPs. The defender's costs for HPs: $c_h(1) = 4$ and $c_h(3) = 1$. The attacker's attack actions a_1 and a_3 with costs $c_1 = 8$ and $c_3 = 4$, resp.; rewards $r_1 = 40$ and $r_3 = 10$, resp.; and success probabilities $p_1 = 0.2$ and $p_3 = 0.4$. In the chance nodes (except the one in the root) nature chooses weather the previous action: interacts with the HP (h), did not interact with HP and was successful (s) or failed (f) with given probabilities at the labels.

where each host t contains x_t real hosts and y_t HPs. Let Z be the set of all networks created as fusion of $x \in X$ with $y \in Y$. We also define $c_h : T \to \mathbb{R}_+$ to be the cost that the attacker pays for adding a HP of type t.

Following our example in Fig. 2, the defender adds $k = 1$ HP which defines her set of actions $Y = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$. Extending each network $x \in X$ by every choice from Y results in $|Z| = 9$ different networks.

3.3 Attacker

The attacker observes the number of hosts of each type, but not whether they are real or HPs. Imperfect observation of the attacker is modeled using *information sets* $\mathcal{I}_a$ that form a partition over the networks. Networks in an information set are indistinguishable for the attacker. Two networks $z = (x, y)$ and $z' = (x', y')$ belong to the same information set $I_a \in \mathcal{I}_a$ if and only if $\forall t \in T : x_t + y_t = x'_t + y'_t$.

In the example in Fig. 2, the attacker observes 6 different information sets, three singletons (containing only one network) {((2,0,1),(3,0,1)), ((1,1,1),(0,1,0)), ((1,0,2),(0,0,1))}) and three information sets containing two networks (denoted with dashed lines). For each networks in information set $I_a \in \mathbb{I}_a$ we create an attack graph which is then translated into a *Markov Decision Process* as described in [Durkota *et al.*, 2015]. The states in the MDP represent the state of the attack (performed actions, achieved facts, etc.) and the set of actions for each state correspond to the set of valid actions in that state defined by the attack graph. A valid policy for the MDP corresponds to an attack policy for the given attack graph. However, there is an exponential number policies for MDP. We define set of attack policies S_{I_a} that the attacker can follow based on attack graph for the networks in I_a.

In the example we show part of the MDP for the information set $(2, 0, 2)$. The attack graph for the network from Fig. 1 with 4 host types contains more than 12 actions, which produces 419845 valid attack policies.

3.4 Player Utilities

The player *utilities* in terminal nodes are computed based on three components: R - the sum of the rewards r_t that the attacker obtained for successfully compromising type t along the history that leads to the terminal node, C - sum of the attacker's costs c_a for the performed actions along that history, and H - the defender's cost for allocating the HPs along that history. The defender's utility is expressed as $u_d = -R - H$ and attacker's utility is $u_a = R - C$.

We demonstrate the player's utility computations for the game instance where nature creates network $x = (2, 0, 1)$, defender allocates a HP to $y = (0, 0, 1)$ and the attacker performs attack policy containing two actions: action a_3 which fails (f) followed by successfully performed action a_1, and the game terminates in the boldface node with defender's utility -41 and attacker's utility 28. The three components are as follows: $R = r_1 = 40$ (only action a_1 succeeded), $C = c_1 + c_3 = 12$ (attempted actions a_1 and a_3) and $H = c_h(3) = 1$ (for allocating HP in as type $t = 3$); thus the attacker's utility is $u_a = R - C = 28$ and the defender's $u_d = -R - H = -41$. Notice that the reward R can be viewed as a zero-sum component (the more the attacker gains the more the defender looses).

3.5 Solution Concepts

Now we formally define the Stackelberg solution concept, where the leader (the *defender* in our case) commits to a publicly known strategy and the follower (the *attacker* in our case) plays a best response to the strategy of the leader. We follow the standard assumption of breaking the ties in favor of the leader (often termed as *Strong Stackelberg Equilibrium*, (SSE); e.g. [Conitzer and Sandholm, 2006; Tambe, 2011]).

We follow the standard definition of strategies in extensive-form games. A *pure strategy* $\pi_i \in \Pi_i$ for player i is an action prescription for every information set in the game (Π_i denotes the set of all pure strategies). *Mixed strategy* $\sigma_i \in \Sigma_i$ for player i is a probability distribution over the pure strategies and Σ_i is the set of all mixed strategies. We overload the notation for the utility function and use $u_i(\sigma_i, \sigma_{-i})$ to denote the expected utility for player i if the players are following the strategies in $\sigma = (\sigma_i, \sigma_{-i})$. *Best response* pure strategy for player i against the strategy of the opponent σ_{-i}, denoted $BR_i(\sigma_{-i}) \in \Pi_i$, is such that $\forall \sigma_i \in \Sigma_i : u_i(\sigma_i, \sigma_{-i}) \leq u_i(BR_i(\sigma_{-i}), \sigma_{-i})$ Now Stackelberg equilibrium is a strategy profile

$$(\sigma_d, \pi_a) = \underset{\sigma'_d \in \Sigma_d, \pi'_a \in BR_a(\sigma'_d)}{\arg\max} u_d(\sigma'_d, \pi'_a)$$

There are several algorithms for computing SSE. For normal-form games, the baseline algorithm is to use multiple linear programs, one for each pure strategy of the follower, assuming to be the best response, and selecting the strategy

of the leader in the one that maximizes the expected utility of the leader [Conitzer and Sandholm, 2006]. This algorithm was later improved to a single linear program of a quadratic size by allowing commitments to correlated strategies, which coincides with standard SSE in normal-form games [Conitzer and Korzhyk, 2011]. The problem of computing SSE is generally NP-hard for extensive-form games [Letchford and Conitzer, 2010] and the baseline algorithm is to use mixed-integer linear program and the sequence-form representation of strategies [Bošanský and Čermák, 2015].

Our HSG, however, have a specific structure that allows us to exploit the algorithm for computing SSE in normal-form games using a single linear program (LP). In Section 4 we use this LP to effectively calculate the upper bound on expected utility of the leader.

4 Game Approximations and Algorithms

In this section we present a collection of *approximated games* for Stackelberg deception games that allow us to apply algorithms that solve the games faster. The approximations are based on relaxing certain aspects of the original game.

4.1 Perfect-Information Game Approximation

A straightforward game approximation is to remove the attacker's uncertainty about the nature's and defender's action, which results in a perfect-information game. In [Durkota *et al.*, 2015] the authors propose an algorithm to solve a game which corresponds to the subgame after the nature's move. We use their algorithm to solve each of the subgames after the nature move and combine their best strategies to construct an optimal strategy for the defender, to which we refer as PI.

4.2 Zero-sum Imperfect-Information

In [Korzhyk *et al.*, 2011] the authors show that under certain conditions approximating the non-zero-sum game as a zero-sum game, which can be solved in polynomial time w.r.t. the size of the game-tree, can provide an optimal strategy for the original game. In the following approximations we use a similar idea for constructing zero-sum game approximations.

Recall that in our game the defender's utility is $u_d = -R-H$ and the attacker's utility is $u_a = R - C$. Our game has a payoff structure where R is the zero-sum component in the utilities and the smaller the difference $|H-C|$ in the outcomes, the closer our game become a zero-sum game. We propose several variants of *zero-sum* game approximations, where one or both players utilities' are modified to satisfy the zero-sum condition $u_d = -u_a$.

We focused on the following four zero-sum setting: both players consider only the expected rewards from the attack policy (ZS1) $u_d = -R$, only attacker's utilities are considered (ZS2) $u_d = -R + C$, only defender's utilities are considered (ZS3) $u_d = -R-H$, and the defender keeps his original utility with adversarial motivation to harm the attacker (ZS4) $u_d = -R - H + C$.

When solving the zero-sum approximation games, we avoid generating exponential number of attacker's attack policies by using Single-Oracle (SO) algorithm, introduced in [McMahan *et al.*, 2003]. It is often used when one player's action space is very large (in our case the attacker's). SO overcomes this difficulty by introducing a *restricted game*, which contains only a subset of the player actions. It iteratively extends the restricted game until it contains an equilibrium. In zero-sum EFG games it was shown in [Jain *et al.*, 2011] that this approach can be used to find NE, which is equivalent to SSE in zero-sum games. For non-zero-sum games no such guarantee holds; however, it may find a sub-optimal solution fast. Thus, we use SO algorithm to solve zero-sum approximations (referred as to ZS1-4) as well as the original game, to which we refer as to SOGS.

Each SO iteration consists of two steps. (i) First, the SSE of the restricted game is computed; for zero-sum games we use LP to find SSE and for general-sum the *mixed integer linear program* (MILP) [Bošanský and Čermák, 2015]. (ii) Second, the attacker computes best response to the defender's strategy from SSE. We compute the attacker's best response for each of his information set separately. The probability of each network in the information set is derived from the defender's strategy, according to which we compute the attacker's optimal attack strategy. The attack graphs for the network in an information set have the same structure, but the actions may have different probabilities of interacting with the honeypots, depending on the defender's action. In [Durkota *et al.*, 2015] the authors present a MDP approach with several pruning techniques for computing the optimal attack policy for a single attack graph. We extend their approach by translating the problem of computing optimal attack policy for the set of attack graphs with the prior probability distribution into partially observable MDP (POMDPs). The attacker has a *belief* about the set of attack graphs he computes optimal attack policy for. We compute optimal attack policy with value iteration algorithm on POMDPs and use Bayes rule to update the attacker's belief about the attack graphs due to action's different probabilities of interacting with the honeypots.

Due to the space limit, we present an example to show the intuition behind the update rule. Assume that the attacker acts in information set z_1 from Fig. 2 and his prior belief of being in left and right state of the game is (a, b), respectively, where $a + b = 1$. Performing action a_3 has 0.5 probability of touching honeypot in the left state and zero in the right state of the game. Thus, if action is performed successfully and does not interact with a honeypot, we update the attacker's belief to $(\frac{0.2a}{0.2a+0.4b}, \frac{0.4b}{0.2a+0.4b})$, where numerator is the probability of reaching the state and the denominator is a normalization factor. Update belief according to this rule for every attacker's action. While computing the optimal attack policy, we also use similar pruning techniques as described in [Durkota *et al.*, 2015].

4.3 Single LP approximation

The authors in [Conitzer and Korzhyk, 2011] present a single linear program (LP) for computing the SSE for a normal-form game (NFG). The LP finds the probability distribution over the outcomes with maximal utility for the defender under the condition that the attacker plays a best response (does not have incentive to deviate). We exploit our extensive-form game's structure which allows us to write the game as a collection of normal-form games and formulate the problem as a single LP. We demonstrate our approach on the example in

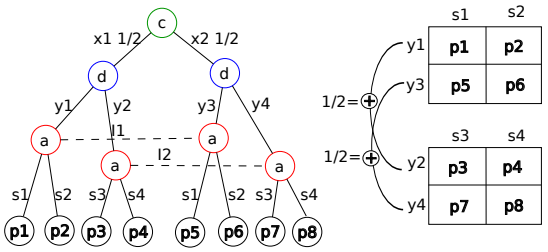


Figure 3: Illustration of extensive-form game translation into set of normal-form games

Fig. 3. For each of the attacker's information sets we construct a normal-form game where the defender chooses an action that leads to the corresponding information set and the attacker chooses an attack policy in the that information set. In Fig. 3 in information set $I1$ the defender chooses between actions $y1$ and $y3$ and in $I2$ between actions $y2$ and $y4$. Each terminal node in the EFG corresponds to an outcome in the NFG (p_1 through p_8 in our example). Furthermore, we incorporate nature's actions by adding a constraint that the sum of outcome probabilities for each subgame after nature acts must equal the probability that Nature chooses this subgame. In our example, the probabilities of outcomes p_1, p_2 (defender's action $y1$) and p_3, p_4 (defender's action $y2$) must sum to $1/2$, the probability that nature plays action $x1$. The LP formulation as follows:

$$\max \sum_{x\in X}\sum_{y\in Y}\sum_{s\in S_{x+y}} p(x,y,s)u_d(x,y,s) \tag{1a}$$

$$\text{s.t. } :(\forall I_a \in \mathcal{I}_a, s_1, s_2 \in S_{I_a}) :$$

$$\sum_{(x,y)\in I_a} P(x,y,s_1)u_a(x,y,s_1) \geq \sum_{(x,y)\in I_a} P(x,y,s_1)u_a(x,y,s_2) \tag{1b}$$

$$\sum_{x\in X}\sum_{y\in Y}\sum_{s\in S_{(x,y)}} p(x,y,s) = 1 \tag{1c}$$

$$(\forall x \in X, y \in Y, s \in S_{(x,y)}) : P(x,y,s) \geq 0 \tag{1d}$$

$$(\forall x \in X) : \sum_{y\in Y}\sum_{s\in S_{(x,y)}} p(x,y,s) = \delta_x \tag{1e}$$

where $p(x,y,s)$ is the probability of reaching terminal state of the game where nature plays x, the defender y and the attacker s. The solution of the LP is an upper bound on the defender's expected utility in the SSE for following reason. Our LP maximizes the defender's expected utility under the constraint that the attacker plays best response (which is a constraint for SSE as well) and structural constraints caused by Nature. However, we do enforce the constraint that the attacker plays a pure best-response (single action) in the information set. Therefore, our constraints are weaker than the constraints for SSE. Which implies that this LP does not excluded any SSE. Thus, the value of the LP (referred as to SSEUB) may be used as an upper bound, and can be computed in polynomial time to the size of the game.

However, formulation our game as a single LP requires finding all attacker's attack policies for each of his information sets. As we mentioned earlier, there is exponential number MDP policies for an attack graph. However, we can reduce this number by considering only *rationalizable* (introduced in [Bernheim, 1984]) attack policies. An attack policy is rationalizable if and only if it is attacker's best response to some belief state about the attack graphs. Considering only set of all rationalizable attack policies in the game, we do not exclude any SSE due to the constraint that the attacker plays best-response (which is rationalizable attack policy) in it. Therefore, we first compute the set of *rationalizable* attack policies for each information set and we consider only them in the LP. We find set of rationalizable attack policies for every information set using our POMDPs approach explained in Sec. 4.2. We use standard approach as described in [Russell and Norvig, 2009] together with techniques of elimination of the dominant policies and the some pruning techniques for solving optimal attack policy for a single attack graph.

From the LP results we extract the defender's strategy as a marginal probability for each action: the probability that defender's plays action $y \in Y$ in state $x \in X$ is $\sum_{s\in S_{(x,y)}} P(x,y,s)$. We will refer to this mixed strategy as σ_d^{LP} and to the defender's utility in the strategy profile where the defender plays σ_d^{LP} and the attacker best response strategy to as SLP; formally, SLP= $u_d(\sigma_d^{LP}, BR_a(\sigma_d^{LP}))$.

5 Experiments

We experimentally evaluate and compare the methods we have proposed for honeypot selection game, namely: (i) single linear program (SLP), (ii) single-oracle algorithm for the original general-sum imperfect-information game (SOGS), (iii) single-oracle in zero-sum approximations (ZS1-4), and (iv) perfect-information general-sum game (PI) to the SSE upper bound (SSEUB).

5.1 Networks and Attack Graphs

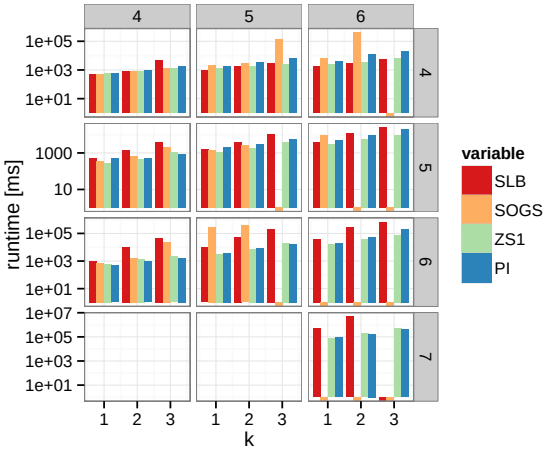
We ran experiments on the *small business* network depicted in Fig. 1 and a *synthetic* network. The attack graph for the business network consists of actions that compromise vulnerabilities in each host. Action success probabilities p_a were generated by the MulVAL tool, while action costs c_a were set randomly in the range from 1 to 100 and rewards for host types r_t between 1000 (for workstations) and 5000 (for a database). The basis network b consists of a server, a database and a workstation (black host types in Fig. 1) and T are all types. The number of types $|T|$, the number of total hosts n and the number of honeypots k are parameters that we alter.

The synthetic networks consist of $|T| = 4$ hosts with direct access to the internet. An attack graph for this network contains one attack action a_t per host type t. The reward for host type t is drawn uniformly from the interval $r_t \in [500, 1500]$, action success probabilities are drawn uniformly from the interval $p_a \in [0.05, 0.95]$, and action costs are drawn uniformly from the interval $c_a \in [1, r_t p_a]$. The basis network b consist of one randomly chosen host type.

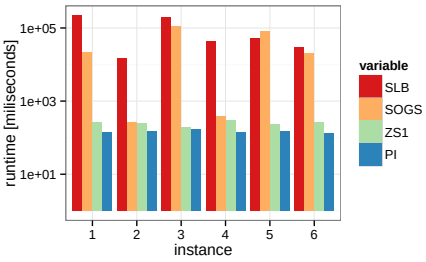
All experiments were run on a 2-core 2.6GHz processor with limitation of 32GB memory and 2 hour of runtime.

5.2 Scalability

In Fig. 4a we present the runtime analysis for the business network. Rows correspond to the number of types $|T|$ in the network, columns to the number of hosts n and the axis-x in



(a)



(b)

Figure 4: Comparison of runtimes on business (a) and synthetic (b) network.

each plot to the number of honeypots k. Experiments that did not finished within 2 hours we plot as runtime of zero.

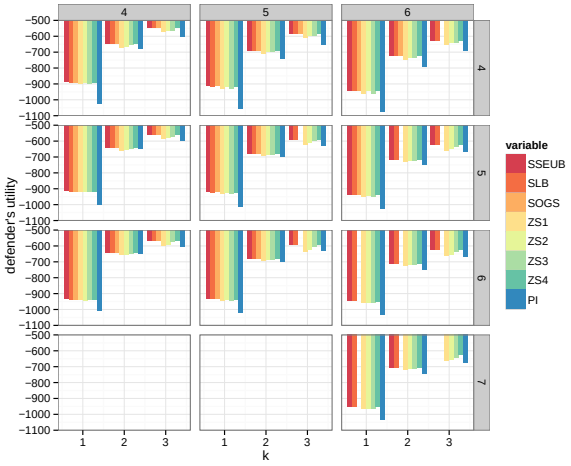
The results show that SLP and SOGS approximations do not scale for larger networks. The bottleneck of SLP approximation is in computing the set of rationalizable actions for every information set. E.g., the game-tree for network with $|T| = 7$ contains thousand of information sets, each with a large number of attack policies and the POMDPs approach could not find the rationalizable actions for each information set in the time limit. SOGS approximation did not scale well due to the rapid growth of the computation time of the MILP after a few single-oracle iterations. In the results, we present only one zero-sum approximation (ZS1), since the others (ZS2-4) had almost identical running time. In Fig. 4b we present computation runtime analysis for the synthetic network. We fixed $T = 4, n = 3, k = 2$ and ran different instances of the games. SSE and SOGS are slowest for similar reason as for business network.

5.3 Defender's Utility

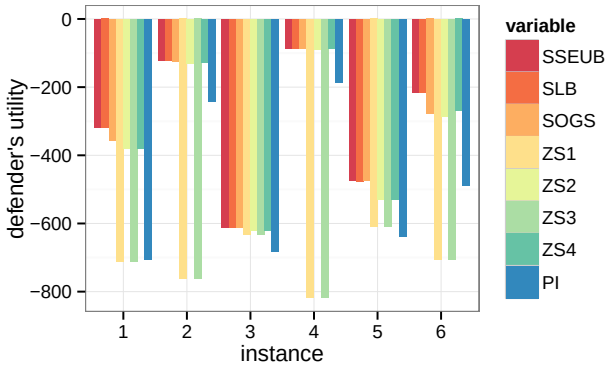
In this section we analyze the quality of the strategies that each approach found. To compare the defender's strategies properly, we compute the attacker's best-responses in the original game. This gives us the defender's worst-case utility. We present the defender's utilities for business network

SLP	SOGS	ZS1	ZS2	ZS3	ZS4	PI
0.104	0.083	2.861	1.951	1.448	0.131	7.767

Table 1: Comparison of relative regrets of the strategies from approximation games.



(a)



(b)

Figure 5: Comparison of leader's utilities for business (a) and synthetic (b) network.

and synthetic network in Fig. 5a and 5b, respectively. The PI game approximation is fast to solve; however, the found strategies are often far from the optimum in the imperfect-information setting, since the defender does not exploit the attacker's limited observations about the network.

For each approximation we compute the upper bounds of the relative regrets as $100(utility - \text{SSEUB})/\text{SSEUB}$, where *utility* is the defender's utility for the approximation. In table 1 we present mean upper bounds of the relative regrets for the business network. The results show that for the business network SOGS approximation finds lowest relative regret and SLP approach has finds second best. However, both of these approaches are not very scalable. While defender's relative regret with strategies found by ZS4 approach are close to the former two approaches and it is much more scalable.

6 Conclusion

We proposed a model to solve the defender's honeypot allocation problem as an imperfect-information game. Finding optimal SSE in such games is a NP-hard problem and state-of-the-art techniques do not scale for larger network. We proposed a collection of approximated games that are solvable in polynomial time (all except SOGS). We ran experiments to analyse each algorithm for solving approximated games and show trade-off between the runtime complexity and quality of the found solution. Game approximations computed with SOGS and SLP have lowest relative regret of the utility for the defender. Third best game approximation was ZS4 which comparing to the former two approximations is scalable for larger networks. We also showed the importance of modeling the element of deception in our game by computing high relative regret of PI approximation.

References

[Ammann *et al.*, 2002] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *CCS*, pages 217–224, 2002.

[Bacic *et al.*, 2006] Eugen Bacic, Michael Froh, and Glen Henderson. Mulval extensions for dynamic asset protection. Technical report, DTIC Document, 2006.

[Bernheim, 1984] B Douglas Bernheim. Rationalizable strategic behavior. *Econometrica: Journal of the Econometric Society*, pages 1007–1028, 1984.

[Boddy *et al.*, 2005] Mark S Boddy, Johnathan Gohde, Thomas Haigh, and Steven A Harp. Course of action generation for cyber security using classical planning. In *ICAPS*, pages 12–21, 2005.

[Bošanský and Čermák, 2015] Branislav Bošanský and Jiří Čermák. Sequence-form algorithm for computing stackelberg equilibria in extensive-form games. In *AAAI*, volume 29, 2015.

[Conitzer and Korzhyk, 2011] Vincent Conitzer and Dmytro Korzhyk. Commitment to correlated strategies. In *AAAI*, 2011.

[Conitzer and Sandholm, 2006] Vincent Conitzer and Tuomas Sandholm. Computing the optimal strategy to commit to. In *EC*, pages 82–90, 2006.

[Durkota *et al.*, 2015] Karel Durkota, Viliam Lisý, Bošanský Branislav, and Christopher Kiekintveld. Optimal network security hardening using attack graph games. In *IJCAI*. AAAI, 2015.

[Grimes *et al.*, 2005] Roger A Grimes, Alexzander Nepomnjashiy, and Jacco Tunnissen. Honeypots for windows. 2005.

[Homer *et al.*, 2013] John Homer, Su Zhang, Xinming Ou, David Schmidt, Yanhui Du, S Raj Rajagopalan, and Anoop Singhal. Aggregating vulnerability metrics in enterprise networks using attack graphs. *Journal of Computer Security*, pages 561–597, 2013.

[Ingols *et al.*, 2006] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *ACSAC*, pages 121–130, 2006.

[Jain *et al.*, 2011] Manish Jain, Dmytro Korzhyk, Ondřej Vaněk, Vincent Conitzer, Michal Pěchouček, and Milind Tambe. A double oracle algorithm for zero-sum security games on graphs. In *AAMAS*, pages 327–334, 2011.

[Korzhyk *et al.*, 2011] Dmytro Korzhyk, Zhengyu Yin, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. Stackelberg vs. nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. *Journal of Artificial Intelligence Research*, 41(2):297–327, 2011.

[Letchford and Conitzer, 2010] Joshua Letchford and Vincent Conitzer. Computing optimal strategies to commit to in extensive-form games. In *ACM EC*, pages 83–92, 2010.

[Letchford and Vorobeychik, 2013] Joshua Letchford and Yevgeniy Vorobeychik. Optimal interdiction of attack plans. In *AAMAS*, pages 199–206, 2013.

[McMahan *et al.*, 2003] H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *ICML*, pages 536–543, 2003.

[Mell *et al.*, 2006] Peter Mell, Karen Scarfone, and Sasha Romanosky. Common vulnerability scoring system. *Security & Privacy*, pages 85–89, 2006.

[Noel and Jajodia, 2004] Steven Noel and Sushil Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *ACM workshop on Visualization and data mining for computer security*, pages 109–118. ACM, 2004.

[Noel and Jajodia, 2008] Steven Noel and Sushil Jajodia. Optimal ids sensor placement and alert prioritization using attack graphs. *Journal of Network and Systems Management*, pages 259–275, 2008.

[Noel *et al.*, 2010] Steven Noel, Sushil Jajodia, Lingyu Wang, and Anoop Singhal. Measuring security risk of networks using attack graphs. *International Journal of Next-Generation Computing*, 1(1):135–147, 2010.

[Obes *et al.*, 2013] Jorge Lucangeli Obes, Carlos Sarraute, and Gerardo Richarte. Attack planning in the real world. *arXiv preprint arXiv:1306.4044*, 2013.

[Ou *et al.*, 2005] Xinming Ou, Sudhakar Govindavajhala, and Andrew W Appel. Mulval: A logic-based network security analyzer. In *USENIX Security*, 2005.

[Ou *et al.*, 2006] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to attack graph generation. In *CCS*, pages 336–345, 2006.

[Píbil *et al.*, 2012] Radek Píbil, Viliam Lisý, Christopher Kiekintveld, Branislav Bošanský, and Michal Pěchouček. Game theoretic model of strategic honeypot selection in computer networks. In *GameSec*, pages 201–220. Springer, 2012.

[Provos, 2004] Niels Provos. A virtual honeypot framework. In *USENIX Security Symposium*, volume 173, 2004.

[Qassrawi and Hongli, 2010] Mahmoud T Qassrawi and Zhang Hongli. Deception methodology in virtual honeypots. In *NSWCTC*, volume 2, pages 462–467. IEEE, 2010.

[Russell and Norvig, 2009] Stuart Russell and Peter Norvig. Artificial intelligence: A modern approach. 2009.

[Sawilla and Ou, 2008] Reginald E. Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. In *Computer Security - ESORICS 2008*, volume 5283, pages 18–34. Springer Berlin Heidelberg, 2008.

[Sheyner *et al.*, 2002] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. In *IEEE S&P*, pages 273–284, 2002.

[Tambe, 2011] Milind Tambe. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press, 2011.