Approximate Solutions for Attack Graph Games with Imperfect Information

Karel Durkota[1], Viliam Lisý[1,2], Branislav Bošanský[3], Christopher Kiekintveld[4]

[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 study the problem of network security hardening, in which a network administrator decides what security measures to use to best improve the security of the network. Specifically, we focus on deploying decoy services or hosts called honeypots. We model the problem as a general-sum extensive-form game with imperfect information and seek a solution in the form of Stackelberg Equilibrium. The defender seeks the optimal randomized honeypot deployment in a specific computer network, while the attacker chooses the best response as a contingency attack policy from a library of possible attacks compactly represented by attack graphs. Computing an exact Stackelberg Equilibrium using standard mixed-integer linear programming has a limited scalability in this game. We propose a set of approximate solution methods and analyze the trade-off between the computation time and the quality of the strategies calculated.

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 [30]. 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 [29, 13]. Believable HPs, however, are costly to set up and maintain. Moreover, a well-informed attacker anticipates the use of HPs and tries to avoid them. To capture the strategic interactions, we model the problem of deciding which services to deploy as honeypots using a game-theoretic framework.

Our game-theoretic model is motivated in part by the success of Stackelberg models used in the physical security domains [33]. One challenge in network security domains is to efficiently represent the complex space of possible attack strategies, we make use of a compact representation of strategies for attacking computer networks called *attack*

graphs. Some recent game-theoretic models have also used attack graphs [19, 12], but these models had unrealistic assumptions that the attacker has perfect information about the original network structure. The major new feature we introduce here is the ability to model the imperfect information that the attacker has about the original network (i.e., the network structure before it is modified by adding honeypots). Imperfect information of the attacker about the network have been proposed before [8, 28], however, the existing models use very abstract one step attack actions which do not allow the rich analysis of the impact of honeypots on attacker's decision making presented here.

Attack graphs (AGs) compactly represent a rich space of sequential attacks for compromising a specific computer network. AGs can be automatically generated based on known vulnerability databases [15, 26] and they are used in the network security to identify the minimal subset of vulnerabilities/sensors to be fixed/placed to prevent all known attacks [32, 24], or to calculate security risk measures (e.g., the probability of a successful attack) [25, 14]. We use AGs as a compact representation of an attack plan library, from which the rational attacker chooses the optimal contingency plan.

The defender in our model selects which types of fake services or hosts to add to the network as honeypots in order to minimize the trade-off between the costs for deploying HPs and reducing the probability of successful attacks. We assume that the attacker knows the overall number of HPs, but does not know which types of services the defender actually allocated as HPs. This is in contrast to previous work [12], where the authors assumed a simplified version to our game, where the attacker knows the types of services containing HPs. The uncertainty in the existing model is only about which specific service/computer is real among the services/computers of the same type. Our model captures more general (and realistic) assumptions about the knowledge attackers have when planning attacks, and we show that the previous perfect information assumptions can lead to significantly lower solution quality.

Generalizing the network hardening models to include imperfect information greatly increases the computational challenge in solving the models, since the models must now consider the space of all networks the attacker believes are possible, which can grow exponentially. Computing Stackelberg equilibria with stochastic events and imperfect information is generally NP-hard [18] and algorithms that compute the optimal solution in this class of games typically do not scale to real-world settings [7]. Therefore we (1) present a novel collection of polynomial time algorithms that compute approximate solutions by relaxing certain aspects of the game, (2) experimentally show that the strategies computed in the approximated models are often very close to the optimal strategies in the original model, and (3) propose novel algorithms to compute upper bounds on the expected utility of the defender in the original game to allow the evaluation of the strategies computed by the approximate models even in large games.

2 Background and Definitions

We define a computer 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, have the same vulnerabilities and connectivity in the network and have the same value for the players (i.e., a collection of identical workstations is modeled as a single type). All hosts of the

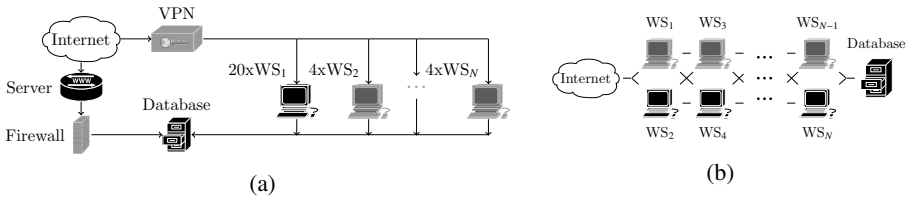


Fig. 1: Simple (a) business-like and (b) chain network topology.

same type present the same attack surface, 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 example network instance is depicted in Fig. 1a, where, e.g., host type WS_1 represents 20 workstations of the same type. We first define attack graphs for the case where attackers have perfect information about the network.

Attack Graph There are multiple representations of AGs common in the literature. *Dependency AGs* are more compact and allow more efficient analysis than the alternatives [21]. Formally, they are a directed AND/OR graph consisting of fact nodes and action nodes. Fact nodes represent logical statements about the network and action nodes correspond to the attacker's atomic actions. Every action a has *preconditions*, set of facts that must be true in order to preform the action, and *effects*, set of facts that become true if action succeeds, in which case the attacker obtains the rewards of corresponding facts. Moreover, action a has probability of being performed successfully $p_a \in [0, 1]$, cost $c_a \in \mathbb{R}^+$ that the attacker pays regardless whether the action succeeded 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 $t \in T$, a specific host of that type is selected with a uniform probability. Since we assume a rational attacker, future actions on the same host type interact with the same host. Interacting with different host of the same type (1) has no additional benefit for the attacker as rewards are defined based on the types and (2) can only increases the probability of interacting with a honeypot and ending the game. The attacker can terminate the attack any time. We use the common monotonicity assumption [1, 26, 23] that once a fact becomes true during an attack, it can never become false again as an effect of any action.

AGs can be automatically generated using various tools. We use the MulVAL [27] to construct dependency AGs from information automatically collected using network scanning tools, such as Nessus[1] or OpenVAS[2]. These AGs consist of an attacker's atomic actions, e.g., exploit actions for each vulnerability of each host, pivoting "hop" actions between the hosts that are reachable from each other, etc. Previous works (e.g., [31]) show that probabilistic metrics can be extracted from the Common Vulnerability Scoring System [22], National Vulnerability Database [2], historical data, red team exercises, or be directly specified by the network administrator.

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

Attack Policy In order to fully characterize the attacker's attack, for a given AG we compute a *contingent attack policy* (AP), which defines an action from the set of applicable actions according to the AG for each situation that may arise during an attack. This plan specifies 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., [21, 5]) are not sufficient as they cannot represent attacker's behavior after action failures. The *optimal AP* is the AP with maximal expected reward for the attacker. See [12] for more details on the attack graphs and attack policies and explanatory examples.

3 Imperfect Information HP Allocation Game

A real attacker does not know the network topology deployed in the company, but may have prior beliefs about the set of networks that the organization would realistically deploy. We assume that the attacker's prior belief about the set of networks that the organization is likely to deploy is common knowledge to both players. However, the attacker may know a subset of host types used by the organization, we refer to as a *basis* of a network, e.g., server, workstation, etc. To capture the set of networks we model the game as an extensive-form game with a specific structure. *Nature* selects a network from the set of possible networks (extensions of the basis network) with the probabilities corresponding to the prior attacker's beliefs about the likelihood of the different networks. The defender observes the actual network and hardens it by adding honeypots to it. Different networks selected by nature and hardened by the defender may lead to networks that look identical to the attacker. The attacker observes the network resulting from the choices of both, nature and the defender, and attacks it optimally based on the attack graph for the observed network. We explain each stage of this three stage game in more detail for the simple example in Fig. 2.

3.1 Nature Actions

For the set of host types T, total number of hosts $n \in \mathbb{N}$ and basis network $b \in \mathbb{N}_0^T$, we generate set of possible networks X including all possible combinations of assigning n hosts into T host types that contain basis in it ($\forall x \in X : \forall t \in T : x_t \geq b_t$). E.g., in Fig. 2 the set of types is $T = \{D, W, S\}$ (e.g., database, workstation, server), and the network basis is $b = (1, 0, 1)$, a database and a server. Nature selects a network $x \in X = \{(2, 0, 1), (1, 1, 1), (1, 0, 2)\}$ with uniform probability $\delta_x = \frac{1}{3}$.

3.2 Defender's Actions

Each network $x \in X$ the defender further extends by adding k honeypots of types from T. Formally, set of all defender's actions is $Y = \{y \in \mathbb{N}_0^T | \sum_{t \in T} y_t = k\}$. Performing action $y \in Y$ on network $x \in X$ results in network $z = (x, y)$, where each host type t consist of x_t real hosts and y_t HPs. The attacker's action on host type t interacts with a honeypot with probability $h_t = \frac{y_t}{x_t + y_t}$. Let $Z = X \times Y$ be the set of all networks created as fusion of $x \in X$ with $y \in Y$. We also define $c_t^h \in \mathbb{R}_+$ to be the cost that the defender pays

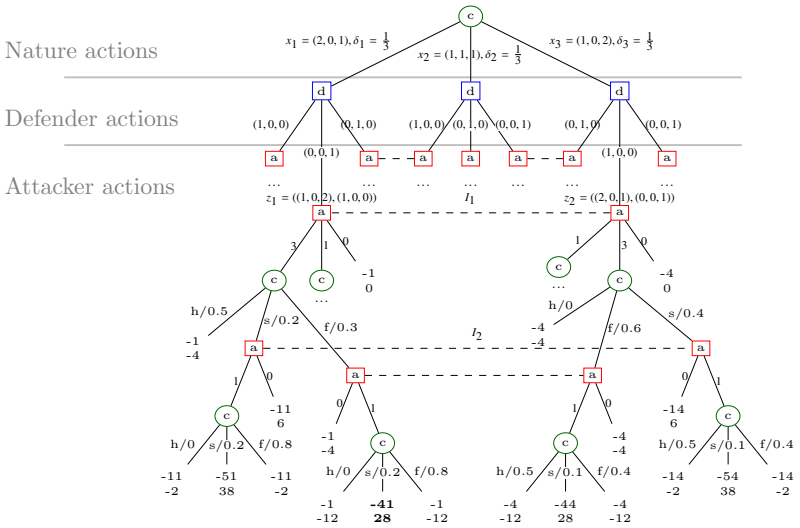


Fig. 2: Simple game tree with $|T| = 3$ host types, basis $b = (1, 0, 1)$, number of hosts $n = 3$ and $k = 1$ HP. The defender's costs for HPs are $c_1^h = 4$ and $c_3^h = 1$. The attacker's attack action 1 (resp. 3) exploits vulnerability of host type 1 (resp. 3), costs $c_1 = 8$ (resp. $c_3 = 4$); reward is $r_1 = 40$ (resp. $r_3 = 10$); and success probability $p_1 = 0.2$ (resp. $p_3 = 0.4$). The action's probabilities of interacting with honeypot (h) depend on defender's honeypot allocations and probabilities of succeeding (s) and failing (f) are accordingly normalized. Attacker's action 0 denotes the attacker ends his attack, which leads to the terminal state. 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 succeeded (s) or failed (f) with the given probabilities.

for adding and maintaining a HP of type t. In the example in Fig. 2 the defender adds $k = 1$ HP and set of the defender's actions is $Y = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$. Extending each network $x \in X$ by every choice from Y results in $|Z| = 9$ different networks.

3.3 Attacker's Actions

The attacker observes the number of hosts of each type, but not whether they are real or honeypots. The attacker's imperfect observation is modeled using *information sets* $\mathcal{I}$ that form a partition over the networks in Z. 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 $I \in \mathcal{I}$ if and only if $\forall t \in T : x_t + y_t = x'_t + y'_t$ holds. Networks $z, z' \in I$ have the same attack graph structure and differ only in the success probabilities and probabilities of interacting with a honeypot, therefore, they produce the same set of attack policies. Let S_I denote the set of valid attack policies in information set I. We also define $I(z)$ (resp. $I(x, y)$) to be a function that for a given network z (resp. (x, y)) returns the information set $I \in \mathcal{I}$ such that $z \in I$ (resp. $(x, y) \in I$). Executing the AP $s \in S_I$ leads to the terminal state of the game. In the example in Fig. 2, the attacker observes 6 different information sets, three singletons (contain only one network), e.g.,

$\{((2,0,1),(1,0,0))\}$, and three information sets that contain two networks (denoted with dashed lines), e.g., $I_1 = \{z_1 = ((2,0,1),(0,0,1)), z_2 = ((1,0,2),(1,0,0))\}$. An example of AP is: perform action 3 in I_1; if it succeeds, continue with action 1 and if fails then 0.

3.4 Players' Utilities

The players *utilities* in terminal state $l \in L$ with path P from the root of the game tree to l is computed based on three components: R_l - the sum of the rewards $\sum_{t \in T^s} r_t$ for successfully compromising host types $T^s \subseteq T$ along P; C_l - the sum of the performed action costs by the attacker along P, and H_l - the defender's cost for allocating the HPs along P. The defender's utility is then $u_\mathrm{d}(l) = -R_l - H_l$ and attacker's utility is $u_\mathrm{a}(l) = R_l - C_l$. Utility for an attack policy is expected utility of the terminal states. Although we assume that R_l is a zero-sum component in the utility, due to player private costs H_l and C_l the game is general-sum.

In our example in Fig. 2, utilities are at the leaf of the game tree labeled with two values. The value at the top is the defender's utility and at the bottom is the attacker's utility in that terminal state. We demonstrate the player's utility computations for the terminal state, the bold one in Fig. 2, we refer as to l_1. The three components are as follows: $R_{l_1} = r_1 = 40$ (only action 1 succeeded), $C_{l_1} = c_1 + c_3 = 12$ (attempted actions were 1 and 3) and $H_{l_1} = c^h_3 = 1$ (for allocating HP in as type $t = 3$); thus the attacker's utility is $u_\mathrm{a}(l_1) = R_{l_1} - C_{l_1} = 28$ and the defender's $u_\mathrm{d}(l_1) = -R_{l_1} - H_{l_1} = -41$.

3.5 Solution Concepts

Formally, we 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. The motivated attacker may be aware of the defender's use of game-theoretic approach, in which case the attacker can compute or learn from past experiences the defender's strategy and optimize against it. We follow the standard assumption of breaking the ties in favor of the leader (often termed as *Strong Stackelberg Equilibrium*, (SSE); e.g. [11, 33]).

We follow the standard definition of strategies in extensive-form games. A *pure strategy* $\pi_i \in \Pi_i$ for player $i \in \{\mathrm{d}, \mathrm{a}\}$ is an action selection 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 $\forall \sigma_i \in \Sigma_i : u_i(\sigma_i, \sigma_{-i}) \leq u_i(BR_i(\sigma_{-i}), \sigma_{-i})$. Let d denote the defender and a the attacker, then Stackelberg equilibrium is a strategy profile

$$(\sigma_\mathrm{d}, \pi_\mathrm{a}) = \underset{\sigma'_\mathrm{d} \in \Sigma_\mathrm{d}; \pi'_\mathrm{a} \in BR_\mathrm{a}(\sigma'_\mathrm{d})}{\arg\max} u_d(\sigma'_\mathrm{d}, \pi'_\mathrm{a}).$$

In our game, the defender chooses honeypot types to deploy in each network $x \in X$ and the attacker chooses pure strategy $\pi_\mathrm{a} \in \Pi_\mathrm{a} = \times_{I \in \mathcal{I}} S_I$, an attack policy to follow in each information set $I \in \mathcal{I}$.

4 Game Approximations

The general cases of computing Stackelberg equilibria of imperfect information games with stochastic events is NP-hard [18]. The state-of-the-art algorithm for solving this general class of games uses mixed-integer linear programming and the sequence-form representation of strategies [7]. Our case of attack graph games is also hard because the size of the game tree representation is exponential in natural parameters that characterize the size of a network (number of host types T, number of hosts n, or number of honeypots k), which further limits the scalability of algorithms. We focus here on a collection of *approximations* that find strategies close to SSE in polynomial time w.r.t. the size of the game tree. We present the general idea of several approximation methods first, and discuss the specific details of new algorithms in the next section.

4.1 Perfect Information Game Approximation

A straightforward game approximation is to remove the attacker's uncertainty about the actions of nature and the defender, which results in a perfect information (PI) game. Although the authors in [18] showed that in general the PI game with chance nodes is still NP-hard to solve, the structure of our game allows us to find a solution in polynomial time. The nature acts only once and only at the beginning of game. After nature's move the game is a PI game without chance nodes, which can be solved in polynomial time w.r.t. the size of the game [18]. To solve the separate subgames, we use the algorithm proposed in [12]. It computes the defender's utility for each of the defender's actions followed by attacker's best response. Next, the algorithm selects the best action to be played in each subgame by selecting the action with maximal utility for the defender. In Sec. 5.2 we discuss the algorithm that computes the optimal attack policy.

4.2 Zero-Sum Game Approximation

In [17] the authors showed that under certain conditions approximating the general sum (GS) game as a zero-sum (ZS) game can provide an optimal strategy for the GS game. In this section we use a similar idea for constructing ZS game approximations, for which we compute a NE that coincides with SSE in ZS games. A NE can be found in polynomial time in the size of the game tree using the LP from [16].

Recall that in our game the defender's utility is $u_{\mathrm{d}}(l) = -R_l - H_l$ and the attacker's utility is $u_{\mathrm{a}}(l) = R_l - C_l$ for terminal state $l \in L$. In the payoff structure R_l is a ZS component and the smaller $|H_l - C_l|$, the closer our game is to a ZS game. We propose four ZS game approximations: (ZS1) players consider only the expected rewards of the attack policy $u_{\mathrm{d}}(l) = -R_l$; (ZS2) consider only the attacker's utility $u_{\mathrm{d}}(l) = -R_l + C_l$; (ZS3) consider only the defender's utility $u_{\mathrm{d}}(l) = -R_l - H_l$; and (ZS4) keep the player's original utilities with motivation to harm the opponent $u_{\mathrm{d}}(l) = -R_l - H_l + C_l$.

We also avoid generating the exponential number of attack policies by using a single oracle algorithm (Sec. 5.1). This algorithm has two subroutines: (i) computing a SSE of a ZS game and (ii) finding the attacker's best response strategy to the defender's strategy. The attacker's best response strategy we find by translating the problem into the *Partially Observable Markov Decision Process* (POMDP), explained in Sec. 5.2.

4.3 Commitment to Correlated Equilibrium

The main motivation for this approximation is the concept of correlated equilibria and an extension of the Stackelberg equilibrium, in which the leader commits to a correlated strategy. It means that the leader not only commits to a mixed strategy but also to signal the follower an action the follower should take such that the follower has no incentive to deviate. This concept has been used in normal-form games [10] and stochastic games [18]. By allowing such a richer set of strategies, the leader can gain at least the same utility as in the standard Stackelberg solution concept.

Unfortunately, computing commitments to correlated strategies is again an NP-hard problem in general extensive-form games with imperfect information and chance nodes (follows from Theorem 1.3 in [34]). Moreover, the improvement of the expected utility value for the leader can be arbitrarily large if commitments to correlated strategies are allowed [18]. On the other hand, we can exploit these ideas and the linear program for computing the Stackelberg equilibrium [10], and modify it for the specific structure of our games. This results in a novel linear program for computing an upper bound on the expected value of the leader in a Stackelberg equilibrium in our game in Section 5.3.

5 Algorithms

5.1 Single Oracle

The single oracle (SO) algorithm is an adaptation of the double oracle algorithm introduced in [6]. It is often used when one player's action space is very large (in our case the attacker's). The SO algorithm uses the concept of a *restricted game* $\hat{G}$, which contains only a subset of the attacker's actions from the full game G.

In iteration m the SO algorithm consists of the following steps: (i) compute SSE strategy profile $(\hat{\sigma}^m_{\mathrm{d}}, \hat{\pi}^m_{\mathrm{a}})$ (if $m = 1$ then $\hat{\sigma}^1_{\mathrm{d}}$ is a strategy where the defender plays every action with uniform probability) of the restricted game $\hat{G}$ and compute the attacker's best response $\pi^m_{\mathrm{a}} = BR_{\mathrm{a}}(\hat{\sigma}^m_{\mathrm{d}})$ in the full game G. If all actions from π^m_{a} are included in the restricted game $\hat{G}$, the algorithm returns strategy profile $(\hat{\sigma}^m_{\mathrm{d}}, \hat{\pi}^m_{\mathrm{a}})$ as a SSE of the full game G. Otherwise, (ii) it extends the restricted game $\hat{G}$ by including the attacker's policies played in $\hat{\pi}^m_{\mathrm{a}}$ and goto (i) with incremented iteration counter m. Initially $\hat{G}$ contains all nature's and the defender's actions and none of the attacker's actions. We use this algorithm to solve all four variants of the ZS approximations proposed in Sec. 4.2. We refer to this approach as SOZS.

The SO algorithm is also well defined for GS games and can be directly applied to the original game. However, it does not guarantee that the computed SSE of the $\hat{G}$ is also the SSE of G. The reason is that the algorithm can converge prematurely and $\hat{G}$ may not contain all the attacker's policies played in SSE in G. Nevertheless, the algorithm may find a good strategy in a short time. We apply this algorithm to our GS game and use *mixed integer linear program* (MILP) formulation ([7]), to compute the SSE of $\hat{G}$ in each iteration. Finding a solution for a MILP is an NP-complete problem, so this algorithm is not polynomial. We refer to this approach as SOGS.

5.2 Attacker's Optimal Attack Policy

The attacker's best response $\pi_a = BR_a(\sigma_d)$ to the defender's strategy σ_d is computed by decomposing the problem into the subproblems of computing the optimal AP for each of the attacker's information set separately. We can do that because subgames of any two informations sets do not interleave (do not have any common state). We calculate the probability distribution of the networks in an information set based on σ_d, which is the attacker's prior belief about the probabilities about the states in the information set. The networks in an information set produce the same attack graph structure. However, the actions may have different probabilities of interacting with the honeypots depending on the defender's honeypot deployment on the path to that network.

Single Network First, we describe an existing algorithm that finds the optimal AP for a single AG for a network, introduced in [12]. The algorithm translates the problem of finding the optimal AP of an AG into a (restricted) finite horizon *Markov Decision Process* (MDP) and uses backward induction to solve it. A state in the MDP is represented by: (i) the set of executable attack actions α in that state (according to the AG), (ii) the set of compromised host types and (iii) the set of host types that the attacker interacted with so far. In each state the attacker can execute an action from α. Each action has a probabilistic outcome of either succeeding (s), failing (f), or interacting with a honeypot (h), described in detail in [12]. After each action, the sets that represent the MDP state are updated based on the AG, the performed action and its outcome (e.g., the actions that became executable are added to α, the performed action and actions no longer needed are are removed, etc.), which represents a new MDP state. If the action successfully compromises a host type t, reward r_t is assigned to that transition. The MDP can be directly incorporated into the game tree, where the attacker chooses an action/transition in each of his states and stochastic events are modeled as chance nodes. The rewards are summed and presented in the terminal states of the game. The authors propose several pruning to generate only promising and needed part of the MDP such as branch and bound and sibling-class theorem and speed-up techniques, such as dynamic programming, which we also adopt.

Multiple Networks The previous algorithm assumes that the MDP states can be perfectly observed. One of our contributions in this paper is an extension of the existing algorithm that finds the optimal AP for a set of networks with a prior probability distribution over them. The attacker has imperfect observation about the networks. We translate the problem into a POMDP. Instead of computing the backward induction algorithm on single MDP, we compute it concurrently in all MDPs, one per network in the information set. In Fig. 2 we show a part of the POMDP for information set I_1, which consists of two MDPs, one for network z_1 and another for z_2.

The same action in different MDPs may have different transition probabilities, so we use Bayes rule to update the probability distribution among the MDPs based on the action probabilities. Let J be the number of MDPs and let $\beta_j(o)$ be the probability that the attacker is in state o in MDP $j \in \{1, \ldots, J\}$. Performing action a leads to state o' with probability $P_j(o, o', a)$. The updated probability of being in j-th MDP given state o' is

$\beta_j(o') = \frac{P_j(o,o',a)\beta_j(o)}{\sum_{j'=1}^{J} P_{j'}(o,o',a)\beta_{j'}(o)}$. This algorithm returns the policy with the highest expected reward given the probability distribution over the networks. During the optimal AP computation, we use similar pruning techniques to those described in [12].

5.3 Linear Program for Upper Bounds

In [10] the authors present a LP that computes SSE of a matrix (or normal-form) game in polynomial time. The LP finds the probability distribution over the outcomes in the matrix with maximal utility for the defender under the condition that the attacker plays a best response. We represent our game as a collection of matrix games, one for each of the attacker's IS, and formulate it as a one LP problem.

Formally, for each attacker's information set $I \in \mathcal{I}$ we construct a matrix game M_I where the defender chooses network $z \in I$ (more precisely an action $y \in Y$ that leads to network $z \in I$) and the attacker chooses an AP $s \in S_I$ for information set I. The outcomes in the matrix game coincide with the outcomes in the original extensive-form game. The LP formulation follows:

$$\max \sum_{x\in X}\sum_{y\in Y}\sum_{s\in S_{I(x,y)}} p_{xys} u_{\mathrm{d}}(x,y,s) \tag{1a}$$

$$\text{s.t. } :(\forall I \in \mathcal{I}, s, s' \in S_I) : \sum_{(x,y)\in I} p_{xys} u_{\mathrm{a}}(x,y,s) \geq \sum_{(x,y)\in I} p_{xys} u_{\mathrm{a}}(x,y,s') \tag{1b}$$

$$(\forall x \in X, y \in Y) : \sum_{x\in X}\sum_{y\in Y}\sum_{s\in S_{I(x,y)}} p_{xys} = 1 \tag{1c}$$

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

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

where the only variables are p_{xys}, which can be interpreted as probability that natures play x, the defender plays y and the attacker is recommended to play s. The objective is to maximize the defender's expected utility. Constraint 1b ensures that the attacker is recommended (and therefore plays) best response. It states that deviation from the recommended action s by playing any other action s' does not increase the attacker's expected utility. 1c and 1d are standard probability constraints and 1e restricts the probabilities of the outcomes to be coherent with the probabilities of the chance node.

We demonstrate our approach on game in Fig. 3a. The game consists of two ISs: $I_1 = \{z_{11}, z_{23}\}$ and $I_2 = \{z_{12}, z_{24}\}$ each corresponds to a matrix game in Fig. 3b. The defender's actions y_1 and y_3 lead to I_1 and y_2 and y_4 lead to I_2. The attacker's attack policies are $S_{I_1} = \{s_1, s_2\}$ and $S_{I_2} = \{s_3, s_4\}$. The probabilities of the terminal states of the game tree correspond to the outcome probabilities in the matrix games ($p_{x,y,s}$). Moreover, the probabilities $p_{111}, p_{112}, p_{123}$ and p_{124} sum to δ_1, as they root from nature's action x_1 played with probability δ_1. The same holds for the other IS.

This LP has weaker restrictions on the solution compared to the MILP formulation for SSE [7] since it does not restrict the attacker to play a pure best response strategy. The objective is to maximize the defender's utility, as in the MILP. Therefore, it does

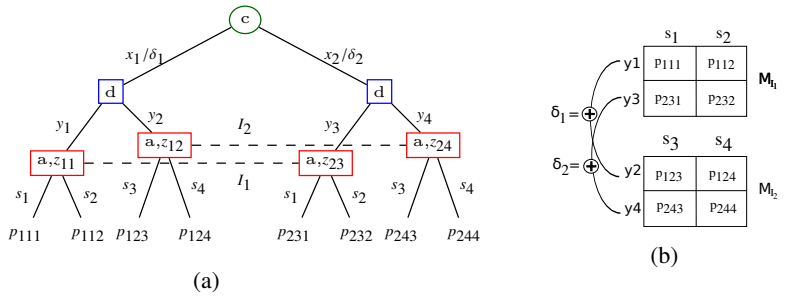


Fig. 3: The extensive-form game in (a) translated into two normal-form games in (b).

not exclude any SSE of the game. The value of this LP, referred to as SSEUB, is an upper bound on the defender's expected utility when playing an SSE.

The drawback of formulating our game as a LP is that it requires finding all (exponential many) AP for each network in advance. We reduce this number by considering only *rationalizable* (in [4]) APs for each information set. An AP is rationalizable if and only if it is the attacker's best response to some belief about the networks in an IS. The set of all rationalizable APs is called *Closed Under Rational Behaviour* (CURB) set [3]. By considering only the CURB set for the attacker, we do not exclude any SSE with the following rationale. Any AP that is in SSE is the set of attacker best responses, so it must be rationalizable and therefore it must be in the CURB set.

From the LP result we extract the defender's strategy as a marginal probability for each defender's action: the probability that defender plays action $y \in Y$ in state $x \in X$ is $\sum_{s \in S_{I(x,y)}} P_{xys}$. We will refer to this mixed strategy as $\sigma_{\mathrm{d}}^{CCE}$ and to the defender's utility in the strategy profile $u_{\mathrm{d}}(\sigma_{\mathrm{d}}^{CSE}, BR_{\mathrm{a}}(\sigma_{\mathrm{d}}^{CSE}))$ as CSE.

CURB for Multiple Networks We further extend the best response algorithms to compute the CURB set. We use the *incremental pruning* algorithm [9], a variant of the backward induction algorithm that in every attacker decision state propagates the CURB set of attack policies for the part of the POMDP that begins in that decision state. Let A be a set of actions in a decision state o. The algorithm is defined recursively as follows. (i) Explore each action $a \in A$ in state o and obtain the CURB set of policies S^a for the part of the POMDP after the action a; (ii) for every action $a \in A$ extend each policy $s_b \in S^a$ to begin with action a in the current state o and then continue with policy s_b; (iii) return the CURB set from the union of all policies $\cup_{a \in A} S_a$ for state o. In step (iii) we use the standard *feasibility linear program* to check whether policy s_b is in the CURB set by finding if there exists a probability distribution between MDPs where s_b yields the highest utility among $\cup_{a \in A} S^a \setminus s_j$, as described in [3, 9].

6 Experiments

We experimentally evaluate and compare our proposed approximation models and the corresponding algorithms. Namely we examine: (i) Perfect information (PI) approxi-

Table 1: Host type values and costs for deploying them as honeypots.

Host type t	database	firewall	WS_n	server
Value of host type r_t	5000	500	1000	2500
Cost for deploying HP of host type c_t^h	100	10	20	50

mation solved with backward induction (Sec. 4.1), (ii) the ZS approximation games solved with SO algorithm, which we refer to as to SOZS1 through SOZS4 (number corresponds to the variant of the ZS approximation), (iii) SO algorithm applied on GS game (SOGS), and (iv) Correlated Stackelberg Equilibrium (CSE) (Sec. 5.3). We also compute the defender's upper bound utility SSEUB and use it as reference point to evaluate the strategies found by the other approximations.

The structure of the experimental section is as follows: in Sec. 6.1 we describe networks we use to generate the action graph game, in Sec. 6.2 we discuss an issue of combinatorially large CURB sets for one of the networks, in Sec. 6.3 we analyze the scalability of the approximated models, in Sec. 6.4 we analyze the quality of the strategies found by the approximated models and in Sec. 6.5 we analyze how the strategies calculated by the approximated models of ZS games depend on how close the games are to being zero-sum. In Sec. 6.6 we investigate the defender's regret for imprecisely modeling the attack graph, and conclude with a case-study analysis in Sec 6.7.

6.1 Networks and Attack Graphs

We use three different computer network topologies. Two of them are depicted in Fig. 1, *small business* (Fig. 1a) and *chain* network (Fig. 1b). Connections between the host types in the network topology correspond to pivoting actions for the attacker in the attack graph (from the compromised host the attacker can further attack the connected host types). We vary the number of vulnerabilities of each host type, which is reflected in the attack graph as an attack action per vulnerability. We generate the actions' success probabilities p_a using the MulVAL that uses Common Vulnerability Scoring System. Action costs c_a are drawn randomly in the range from 1 to 100, and host type values r_t and the cost for honeypot c_t^h of host type t are listed in Table 1. We assume that the more valuable a host type is the more expensive it is to add a HP of that type. We derive honeypot costs linearly from the host values with a factor of 0.02. The basis network b for the business and chain network consists of the black host types in Fig. 1. We scale each network by adding the remaining depicted host types and then by additional workstations. We also scale the total number of hosts n in the network and the number of honeypots k. Each parameter increases combinatorially the size of the game.

The third network topology is the *unstructured* network, where each host type is directly connection only to the internet (not among each other). The attack graph consists of one attack action t per host type T, which attacker can perform at any time. For the unstructured network we create diverse attack graphs by drawing: host types values uniformly from $r_t \in [500, 1500]$, action success probabilities uniformly from $p_t \in [0.05, 0.95]$ and action costs uniformly from $c_t \in [1, r_t p_t]$. We restrict the action costs from above with $r_t p_t$ to avoid the situations where an action is not worth perform-

ing for the attacker, in which case the attack graph can be reduced to a problem with $|T| - 1$ types. The basis network b consists of two randomly chosen host types.

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

6.2 Analytical Approach for CURB for Unstructured Network

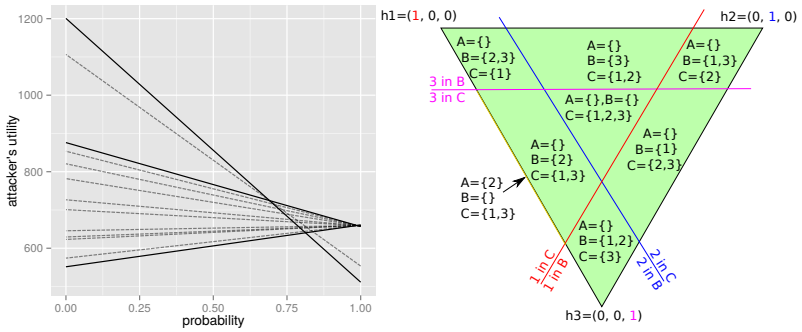
The incremental pruning algorithm described in Sec. 5.3 generates a very large number of attack policies in the CURB set for the unstructured network. In order to be able to compute the upper bound for solution quality for larger game and in order to understand the complexities hidden in CURB computation, we analyze this structure of the curb for this simplest network structure formally. In Fig. 4a we show an example of the attacker's utilities for the policies in a CURB set generated for an information set with two networks. Recall $h_t = \frac{y_t}{x_t+y_t}$ is the probability that action that interacts with host type t (in this case action t) will interact with a honeypot. On the x-axis is probability distribution space between two networks, one with $h_t = 0$ ($y_t = 0$ and $x_t > 0$) and other with $h_t = 1$ ($y_t > 0$ and $x_t = 0$). The y-axis is the attacker's utility for each attack policy in the CURB. The algorithm generates the attack policies known as *zero optimal area policies* (ZAOPs) [20], denoted with dashed lines in the figure. A policy is ZAOP if and only if it is an optimal policy at a single point in the probability space (dashed policies in Fig. 4a). The property of ZAOP is that there is always another policy in the CURB set with strictly larger undominated area. It raises two questions: (i) can we remove ZAOPs from the CURB set and (ii) how to detect them. Recall that in SSE the attacker breaks ties in favour of the defender. Therefore, we can discard ZAOP as long as we keep the best option for the defender.

Further analysis showed that ZAOPs occur when $h_t = 1 - \frac{c_t}{p_t r_t}$ (at probability 0.69 and 0.99 in Fig. 4a). It is because the expected reward of action t at that point is $p_t(1 - h_t)r_t - c_t = p_t r_t \frac{c_t}{p_t r_t} - c_t = 0$, which means that the attacker is indifferent whether to perform action t or not. The algorithm at probability $h_t = 1 - \frac{c_t}{p_t r_t}$ generates the set of attack policies with all possible combinations where the attacker can perform action t in the attack policy. Let $P(t)$ be the probability that the attacker performs action t in an attack policy. The defender's utility for action t is $-r_t p_t(1 - h_t)P(t) = -c_t P(t)$. Because the attacker breaks ties in favour of the defender, at $h_t = 1 - \frac{c_t}{p_t r_t}$ the attacker will choose not to perform action t and we can keep only the policy that does not contain action t.

Furthermore, we categorize each action t based on h_t to one of three classes: to class A if $h_t = 0$, to class B if $0 < h_t < 1 - \frac{c_t}{p_t r_t}$ and to class C if $1 - \frac{c_t}{p_t r_t} < h_t$. In an optimal attack policy: (i) all actions from A are performed first and any of their orderings yield the same expected utility for the attacker, (ii) all actions from B are performed and their order is in increasing order of $\frac{p_t(1-h_t)r_t - c_t}{h_t}$ ratios, and (iii) none of the actions from C are performed, as they yield negative expected reward for the attacker. We partition all probability distributions into regions $h_t = 1 - \frac{c_t}{p_t r_t}$ and in each region we assign actions to the classes. We find the set of optimal attack policies for each region. The attack policies in one region differ from each other in ordering of the actions in B.

In Fig. 4b we show an example of the probability distribution space of three networks in an information set. The probabilities that actions $1, 2$ and 3 interact with a honeypot represent a point (h_1, h_2, h_3). We partition the probability space and assign

each action to a class. In all experiments we use this approach to generate the CURB set without ZAOPs in games for unstructured networks.



(a) (b)

Fig. 4: (a) Attack policies from a CURB set for an information set for the unstructured network. (b) Probability space partitioning by action belonging into the categories.

6.3 Scalability

In this section we compare the runtimes of the algorithms. We present the mean runtimes (x-axis in logarithmic scale) for each algorithm on business (Fig. 5a), chain (Fig. 5b top) and unstructured (Fig. 5b bottom) with of 5 runs (the runtimes were almost the same for each run). We increased the number of host types T, number of hosts n and number of honeypots k. The missing data indicate that the algorithm did not finish within a 2 hour lime limit. From ZS approximations we show only SOZS4 since the others (SOZS1, SOZS2 and SOZS3) had almost identical runtimes.

From the results we see that least scalable are SOGS and CSE approach. SOGS is very slow due to the computation time of the MILP. Surprisingly, in some cases the algorithm solved more complex game (in Fig. 5b $T = 7, n = 7, k = 3$) and not the simpler game (in Fig. 5b $T = 7, n = 7, k = 1$). The reason is that the more complex game requires 3 iterations to converge, while the simpler games required over 5 iterations, after which the restricted game became too large to solve the MILP. The CSE algorithm was the second worst. The bottle-neck is in the incremental pruning algorithm subroutine, which took on average 91% of the total runtime for the business network and 80% for the chain network. In the unstructured network the problem specific CURB computation took only about 4% of total runtime. The algorithms for ZS1-ZS4 and PI approximation showed the best scalability. Further scaling was prohibited due to memory restrictions.

6.4 Solution Quality

In this section we analyze the quality of the strategy that each approximation algorithm found. We use the concept of *relative regret* to capture the relative difference in the defender's utilities for using one strategy instead of another. Formally, the relative regret

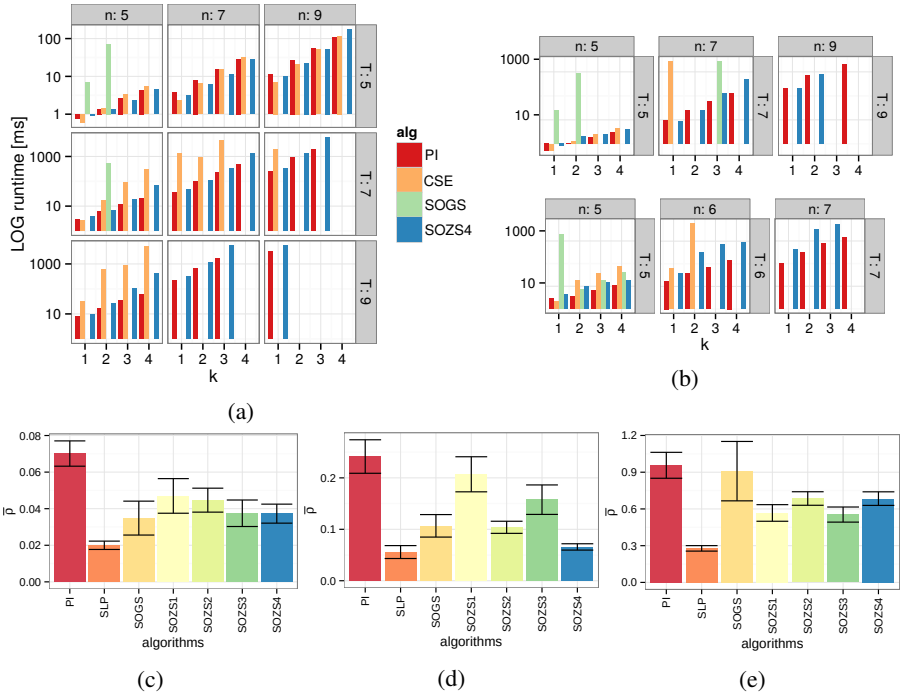
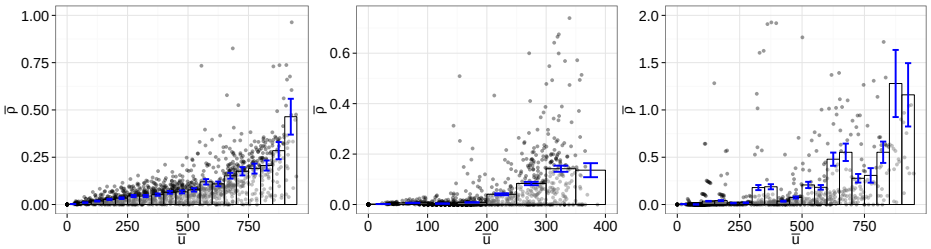


Fig. 5: Comparison of approximations scalability for (a) business, and (b top) chain and (b bottom) unstructured network. In (c),(d) and (e) we compare the defender's upper bound of relative regret of strategies computed with approximation algorithms business, chain and unstructured network, respectively.

of strategy σ_{d} w.r.t. the optimal strategy σ^*_{d} is $\rho(\sigma_{\mathrm{d}}, \sigma^*_{\mathrm{d}}) = \frac{u_{\mathrm{d}}(\sigma^*_{\mathrm{d}}, BR_{\mathrm{a}}(\sigma^*_{\mathrm{d}})) - u_{\mathrm{d}}(\sigma_{\mathrm{d}}, BR_{\mathrm{a}}(\sigma_{\mathrm{d}}))}{u_{\mathrm{d}}(\sigma^*_{\mathrm{d}}, BR_{\mathrm{a}}(\sigma^*_{\mathrm{d}}))}$. The higher the regret $\rho(\sigma_d, \sigma^*_d)$ the worse strategy σ_d is compared to strategy σ^*_d for the defender. We calculate the defender's *upper bound for relative regret* $\bar{\rho}$ by comparing the utilities of the computed strategies to SSEUB. Notice that the results are overestimation of the worst-case relative regrets for the defender. In Fig. 5 we show the means and standard errors $\bar{\rho}$ of 200 runs for the business network (Fig. 5c), chain network (Fig. 5d) and unstructured network (Fig. 5e), with $T = 5, n = 5$ and $k = 2$. In each instance we altered the number of vulnerabilities of the host types and host type values. The action costs c_i we draw randomly from $[1, 100]$ and action success probabilities p_i from $[0, 1]$.

The CSE algorithm computed the best strategies with lowest $\bar{\rho}$. The SOGS is second best in all networks except unstructured. Unfortunately, these algorithms are least scalable. The strategies computed with SOZS algorithm are within reasonable quality. In the business network SOZS4 performed the best among the ZS approximations and in the chain network the computed strategies were almost as good as the best strategies computed with the CSE algorithm. However, in the unstructured network it performed worse. In ZS4 approximations the defender's utility is augmented to prefer outcomes with expensive attack policies for the attacker. Therefore, the ZS4 approximation



(a) (b) (c)

Fig. 6: The defender's relative regret dependence on game *zero-sumness* (computed as average $u_a(l) + u_d(l)$) for (a) business, (b) chain and (c) unstructured networks.

works well for networks where long attack policies are produced. In chain networks the database is the furthest from the internet and in the unstructured network is the closest. PI algorithm computed the worst strategies in all networks. Because of the good tradeoff between scalability and quality of the produces strategies, we decided to further analyze the strategies computed with SOZS4 algorithm.

6.5 Quality of ZS Approximations

The zero sum approximations rely on a zero-sum assumption not actually satisfied in the game. It is natural to expect that the more this assumption is violated in the solved game, the lower the solution quality will be. In order to better understand this tradeoff, we analyze the dependence of the quality of the strategy computed with SOZS4 algorithm on the amount of *zero-sumness* of the original game. We define a game's *zero-sumness* as $\bar{u} = \frac{1}{|L|} \sum_{l \in L} (|u_d(l) + u_a(l)|)$, where L is the set of all terminal states of the game.

In Fig. 6 we show the upper bound for relative regret $\bar{\rho}$ on the y-axis for the strategies computed by SOZS4 and amount of game zero-sumness $\bar{u}$ on the x-axis for 300 randomly generated game instances for the business network (Fig. 6a), chain network (Fig. 6b) and unstructured network (Fig. 6c) with $T = 5, n = 5$ and $k = 2$. In each instance we randomly chose the attacker's action costs $c_a \in [1, 500]$ and honeypot costs $c_t^h \in [0, 0.1 r_t]$, while host type values r_t were fixed. We also show the means and the standard errors of the instances partitioned by step sizes of 50 for $\bar{u}$.

We show that the games with low zero-sumness can be approximated as zero-sum games and the computed strategies have low relative regret for the defender. For example, in a general sum game with $\bar{u} = 250$ the defender computes a strategy at most 6% worse than the optimal strategy.

6.6 Sensitivity Analysis

The defender's optimal strategy depends on the attack graph structure, the action costs, success probabilities and rewards. In real-world scenarios the defender can only estimate these values. We analyze the defender's strategy sensitivity computed with SOZS4 to perturbations in action costs, probabilities and rewards in attack graphs.

We generate the defender's estimate of the attack graph by perturbing the original attack graph actions as follows: (i) action success probability are chosen uniformly from the interval $[p_a - \delta_p, p_a + \delta_p]$ restricted to $[0.05, 0.95]$ to prevent it becoming impossible or infallible, (ii) action costs are chosen uniformly from interval $[c_a(1-\delta_c), c_a(1+\delta_c)]$, and (iii) rewards for host t from uniformly from the interval $[r_t(1-\delta_r), r_t(1+\delta_r)]$, where p_a, c_a and r_t are the original values and δ_p, δ_c and δ_r is the amount of perturbation. The action probabilities are perturbed absolutely (by $\pm\delta_p$), but the costs and rewards are perturbed relative to their original value (by $\pm\delta_c c_a$ and $\pm\delta_r r_f$). The intuition behind this is that the higher the cost or reward values the larger the errors the defender could have made while estimating them, which cannot be assumed for the probabilities.

We compute (i) the defender's strategy σ_d on the game with the original attack graphs and (ii) the defender's strategy σ_d^p on the game with the perturbed attack graph. Fig. 7 presents the dependence of the relative regret $\rho(\sigma_d, \sigma_d^p)$ on the perturbations of each parameter individually ($\delta_p, \delta_c, \delta_r$) and altogether ($\delta_a$). The results suggest that the regret depends significantly on the action success probabilities and the least on the action costs. E.g., the error of 20% ($\delta_a = 0.2$) in the action probabilities results in a strategy with 25% lower expected utility for the defender than the strategy computed based on the true values. The same imprecision in action costs or host type rewards result in only 5% lower utility.

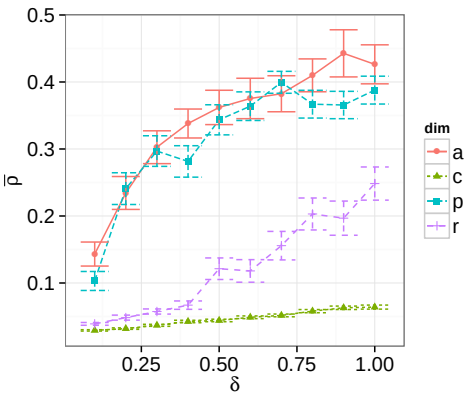


Fig. 7: The defender's utility regret for perturbed action success probabilities, action costs, and host type values.

6.7 Case Study

In order to understand what types of errors individual approximations make, we analyze the differences in strategies computed by the algorithms on a specific game for business network with $T = 5$, $n = 5$ and $k = 2$. The network basis is $b = (1, 0, 1, 0, 1)$, where the elements correspond to the number of databases, firewalls, WS1, WS2 and servers, respectively. There are $|X| = 15$ possible networks, each selected with probability $\delta_x = \frac{1}{15}$. The defender can deploy honeypots in $|Y| = 15$ ways and with honeypot costs as showed in Table 1. There are 225 network settings partitioned into 70 information sets for the attacker. Table 2 presents the comparison of the strategy qualities computed with the algorithms and their runtime in seconds. The upper bound for the defender's optimal utility is SSEUB=-643. The best strategy was computed with CSE algorithm with utility $u_\mathrm{d} = -645$. Although the difference between the utilities is very small, it suggests that the CSE strategy in not necessary optimal. We compare the strategies of the other algorithms to the CSE strategy.

SOGS computed the second best strategy ($u_\mathrm{d} = -654$) and failed to compute the optimal strategy because the restricted game lacks strategies played by attacker in SSE.

Table 2: Algorithm comparison for the case-study.

algorithm	SSEUB	CSE	SOGS	SOZS1	SOZS2	SOZS3	SOZS4	PI
defender's utility	-643	-645	-654	-689	-665	-676	-656	-699
runtime [s]	2.9	3.2	6027	1.3	1.5	1.3	1.5	1.4

For example, both strategies from SOGS and CSE in the network $x_1 = (3, 0, 1, 0, 1)$ play $y_1 = (0, 0, 1, 0, 1)$ (adds a WS1 and a server as HPs) with probability 1. The attacker aims to attack the most valuable host (database) either via WS1 (policy s_1) or server (policy s_2). Both have the same probability of interacting with a honeypot 0.5 and a rational attacker will choose s_2 to compromise the server as well. Attack policy s_2 leads to a terminal state with the defender's expected utility -600. The strategy from CSE, in contrast to strategy from SOGS, additionally plays $y_2 = (1, 0, 0, 0, 1)$ in network $x_2 = (2, 0, 2, 0, 1)$ with probability 0.037, which leads to the same information set as action y_1 in x_1. The attacker's uncertainty between the two states in the information set changes his optimal attack policy from s_2 to s_1 for that information set. Attacking via the WS1 host type has a lower probability of interacting with the HP than via a server, which yields the defender expected utility -538, since the server will not be compromised. The restricted game in SOGS algorithm did not contain strategy s_1, so the computed strategy did not play y_2 in x_2 at all.

The PI strategy has the lowest defender's utility as it does not exploit the attacker's imperfect information at all. In this game the defender always adds a server host type as a honeypot to try to stop the attacker at the beginning. The second honeypot is added by a simple rule: (i) if the database can be compromised only via server and WS1, add honeypot of WS1 host type, otherwise (ii) as a database host type.

SOZS4 computed the best strategy among the ZS approximations. However, each of them have certain drawbacks. In SOZS1 and SOZS2 the defender ignores his costs for deploying the honeypots; these strategies often add database hosts as honeypots, which is in fact the most expensive honeypot to deploy. In SOZS2 and SOZS4 the defender prefers outcomes where the attacker has expensive attack policies. They often deploy honeypots with motivation for the attacker to have an expensive costs for attack policies (e.g., a strategy computed with SOZS2 adds database as a honeypot in 74% while the strategy from CSE only in 43%). Strategies computed with SOZS3 and SOZS4 are difficult to analyze. The strategies often miss the precise probability distribution between the networks where the attacker is indifferent between the attack policies and therefore chooses the one in favour for the defender. There is no general error they make in placing the honeypots as with the previous strategies.

7 Conclusion

We study a class of attack graph games which models the problem of optimally hardening a computer network against a strategic attacker. Previous work on attack graph games has made simplifying assumptions that the attacker has perfect information about the original structure of the network, before any actions are taken to harden the network. We consider the much more realistic case where the attacker only observes the current

network, but is uncertain about how the network has been modified by the defender. We show that modeling imperfect information in this domain has a substantial impact on the optimal strategies for the game.

Unfortunately, modeling the imperfect information in attack graph games leads to even larger and more computationally challenging games. We introduce and evaluate several different approaches for solving these games approximately. One of the most interesting approaches uses a relaxation of the optimal MILP solution method into an LP by removing the constraint that attackers play pure strategies. This results in a polynomial method for finding upper bounds on the defender's utility that are shown to be quite tight in our experiments. We are able to use this upper bound to evaluate the other approximation techniques on relatively large games. For games that are close to zero-sum games, the zero-sum approximations provide a good tradeoff between scalability and solution quality, while the best overall solution quality is given the by the LP relaxation method. Several of these methods should generalize well to other classes of imperfect information games, including other types of security games.

Acknowledgments

This research was supported by the Office of Naval Research Global (grant no. N62909-13-1-N256), the Danish National Research Foundation and the National Science Foundation of China (under the grant 61361136003) for the Sino-Danish Center for the Theory of Interactive Computation. Viliam Lisý is a member of the Czech Chapter of The Honeynet Project.

References

1. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Proc. of CCS. pp. 217–224 (2002)
2. Bacic, E., Froh, M., Henderson, G.: Mulval extensions for dynamic asset protection. Tech. rep., DTIC Document (2006)
3. Benisch, M., Davis, G.B., Sandholm, T.: Algorithms for closed under rational behavior (curb) sets. J. Artif. Int. Res. 38(1), 513–534 (May 2010)
4. Bernheim, B.D.: Rationalizable strategic behavior. Econometrica pp. 1007–1028 (1984)
5. Boddy, M.S., Gohde, J., Haigh, T., Harp, S.A.: Course of action generation for cyber security using classical planning. In: Proc. of ICAPS. pp. 12–21 (2005)
6. Bošanský, B., Kiekintveld, C., Lisý, V., Pěchouček, M.: An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. J. Artif. Int. Res. pp. 829–866 (2014)
7. Bošanský, B., Čermak, J.: Sequence-form algorithm for computing stackelberg equilibria in extensive-form games. In: Proc. of AAAI Conf. on AI. pp. 805–811 (2015)
8. Carroll, T.E., Grosu, D.: A game theoretic investigation of deception in network security. Security and Communication Networks 4(10), 1162–1172 (2011)
9. Cassandra, A., Littman, M.L., Zhang, N.L.: Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In: Proc. of UAI. pp. 54–61. Morgan Kaufmann Publishers Inc. (1997)
10. Conitzer, V., Korzhyk, D.: Commitment to correlated strategies. In: Proc. of AAAI. pp. 632–637 (2011)

11. Conitzer, V., Sandholm, T.: Computing the optimal strategy to commit to. In: Proc. of ACM EC. pp. 82–90. ACM (2006)
12. Durkota, K., Lisý, V., Bošanský, B., Kiekintveld, C.: Optimal network security hardening using attack graph games. In: Proc. of IJCAI. pp. 7–14 (2015)
13. Grimes, R.A., Nepomnjashiy, A., Tunnissen, J.: Honeypots for windows (2005)
14. Homer, J., Zhang, S., Ou, X., Schmidt, D., Du, Y., Rajagopalan, S.R., Singhal, A.: Aggregating vulnerability metrics in enterprise networks using attack graphs. J. Comput. Secur. 21(4), 561–597 (Jul 2013)
15. Ingols, K., Lippmann, R., Piwowarski, K.: Practical attack graph generation for network defense. In: Proc. of ACSAC. pp. 121–130 (2006)
16. Koller, D., Megiddo, N., Von Stengel, B.: Efficient computation of equilibria for extensive two-person games. Games and Economic Behavior 14(2), 247–259 (1996)
17. Korzhyk, D., Yin, Z., Kiekintveld, C., Conitzer, V., Tambe, M.: Stackelberg vs. nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. J. Artif. Int. Res. 41(2), 297–327 (2011)
18. Letchford, J., Conitzer, V.: Computing optimal strategies to commit to in extensive-form games. In: Proc. of ACM EC. pp. 83–92 (2010)
19. Letchford, J., Vorobeychik, Y.: Optimal interdiction of attack plans. In: Proc. of AAMAS. pp. 199–206 (2013)
20. Littman, M.L.: The witness algorithm: Solving partially observable markov decision processes. Tech. rep., Providence, RI, USA (1994)
21. Lucangeli Obes, J., Sarraute, C., Richarte, G.: Attack planning in the real world. In: Working notes of SecArt'2010 at AAAI. pp. 10–17 (2010)
22. Mell, P., Scarfone, K., Romanosky, S.: Common vulnerability scoring system. Security & Privacy pp. 85–89 (2006)
23. Noel, S., Jajodia, S.: Managing attack graph complexity through visual hierarchical aggregation. In: Proc. of ACM VizSEC/DMSEC. pp. 109–118. ACM (2004)
24. Noel, S., Jajodia, S.: Optimal ids sensor placement and alert prioritization using attack graphs. Journal of Network and Systems Management pp. 259–275 (2008)
25. Noel, S., Jajodia, S., Wang, L., Singhal, A.: Measuring security risk of networks using attack graphs. International Journal of Next-Generation Computing 1(1), 135–147 (2010)
26. Ou, X., Boyer, W.F., McQueen, M.A.: A scalable approach to attack graph generation. In: Proc. of ACM CCS. pp. 336–345. ACM (2006)
27. Ou, X., Govindavajhala, S., Appel, A.W.: Mulval: A logic-based network security analyzer. In: Proc. of USENIX SSYM. pp. 113–128. USENIX Association, Berkeley, CA, USA (2005)
28. Píbil, R., Lisý, V., Kiekintveld, C., Bošanský, B., Pěchouček, M.: Game theoretic model of strategic honeypot selection in computer networks. In: In Proc. of GameSec, pp. 201–220. Springer (2012)
29. Provos, N.: A virtual honeypot framework. In: Proc. of USENIX SSYM. pp. 1–14. Berkeley, CA, USA (2004)
30. Qassrawi, M.T., Hongli, Z.: Deception methodology in virtual honeypots. In: Proc. of NSWCTC. vol. 2, pp. 462–467. IEEE (2010)
31. Sawilla, R.E., Ou, X.: Identifying critical attack assets in dependency attack graphs. In: Proc. of ESORICS 2008, vol. 5283, pp. 18–34. Springer Berlin Heidelberg (2008)
32. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: IEEE Symp. Security and privacy. pp. 273–284. IEEE (2002)
33. Tambe, M.: Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned. Cambridge University Press, New York, NY, USA, 1st edn. (2011)
34. Von Stengel, B., Forges, F.: Extensive form correlated equilibrium: definition and computational complexity. Mathematics of Operations Research 33(4), 1002–1022 (2008)