Deception Games in Network Security

Minimal Ph.D. Proposal

Karel Durkota
karel.durkota@agents.fel.cvut.cz
Department of Computer Science, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic

January 5, 2016

.....
Ing. Karel Durkota
Author

.....
Prof. Michal Pěchouček, PhD.
Supervisor

Contents

1 Introduction

Networked computer systems support a wide range of critical functions in both civilian and military domains. Securing this infrastructure is extremely costly using classical defensive mechanism to protect the network, e.g. firewalls, encryption and intrusion detection systems. Moreover, for large networks it is very hard for human network administrator to select best available hardening option, and there is a need for new automated decision support systems that aid human network administrators to detect and prevent attacks.

In this research we focus on network security hardening problems in which a network administrator reduces the risk of attacks on the network by various hardening actions. A particularly interesting action is adding *honeypots* which is a fake host or service added into the network[1] [28]. Legitimate users do not interact with honeypots; hence, honeypots act as decoys and distract attackers from the real hosts, send intrusion detection alarms to the administrator, and/or gather detailed information the attacker's activity [27, 14]. Deciding how to optimally allocate honeypots in the network for the best tradeoff between cost for honeypots and probability of detecting the attackers brings a challenging decision for the administrator.

A good mathematical tool to answer such questions is *game theory*, a framework for studying decision making problems of multiple interacting decision makers, called *players*, in a common environment. We focus on a two-player game: one player, the *defender*, seeks to find the best tradeoff between a cost for allocating the honeypots and an expected loss caused by attacker successfully compromising the network. The second player, the attacker, on the other hand wants to compromise the network avoiding the detection. Game theory is well suited for computing the players' strategies, which is a prescription of an action (or probability distribution over actions) in every reachable situation that the players should perform to maximize their respective expected utilities.

We use game theory to model this interaction and to determine the best way to use honeypots against the attacker. We investigate game-theoretic models of network hardening using honeypots that extends the existing line of Stackelberg security games [37] by combining two elements: (1) adopting a compact representation of strategies for attacking computer networks called *attack graphs*, (2) the defender uses *deception* instead of directly allocating resources to harden targets.

Attack graphs (AGs) can represent a rich space of attacker action sequences for compromising a specific computer network. They can be automatically generated based on known vulnerability databases [16, 24] and they are widely used in network security to identify the minimal subset of vulnerabilities/sensors to be fixed/placed to prevent all known attacks [33, 21], or to calculate security risk measures (e.g., the probability of a successful attack) [22, 15]. We use them as a compact representation of an attack plan library, from which the rational

[1] To honeypots that are not computers are referred as to *honeytoken* which could be a fake file, link or any information in the system.

attacker chooses the optimal contingency attack plan to follow. However, finding the optimal attack plan in an attack graph is an NP-hard problem [13]. We address this issue by translating attack graphs into an MDP and introducing a collection of pruning techniques that reduce the computation considerably.

1.1 Research Objectives

The dissertation aims to improve the state-of-the-art techniques for decision making in network security domain using game-theoretic approach and attack graphs.

(Objective 1) Formalize and define a practical and tractable subset of network security configuration problems.

(Objective 2) Propose a game-theoretic model and algorithms for solving the problems.

(Objective 3) Theoretically and practically analyze the proposed algorithms.

(Objective 4) Empirically evaluate a prototype algorithm on real data.

1.2 Approach

In order to create a network security configuration tool using game theory, first, we studied the attack graph – a compact representation of all *known* possible attack sequences how the attacker could compromise targeted network – as a tool to model the attacker's behaviour. Second, a game-theoretic approach was be formalized for our network security configuration problem as a two-player game between the administrator and an attacker. The game formalization consists of developing (i) set of actions available for each player (at every stage of the game), (ii) information that players have during the game and (iii) players' payoffs for every game outcome. We began with simpler perfect-information game, where both players are well-informed about the state of the network. Afterwards, the game has been extended with player uncertainty about the network structure, attacker targets or even the attacker's level of rationality. We solve games by finding the Strong Stackelberg Equilibrium (SSE) as a solution concept, which is often used in security games. We will further extend game with additional defender's action repertoire and use honeypots in a specific scenario, such as, botnet detection. Finally, scalability and quality of the produced strategies of the proposed algorithms and their extensions will be experimentally analyzed and evaluated on real-world networks and data.

2 Related Work

2.1 Honeypots

Classical network security techniques are introducing firewalls, encryption and intrusion detection systems (IDS). A particularly interesting technique is in-

troducing honeypots into the network. Honeypots ([36]) can be classified by level of interaction they provide to deceive the attacker. Low-interaction honeypots only emulates the computer or service; while they are easy to deploy, there is limited type of information they can capture and they are easier to detect by the attackers. High-interaction honeypots contain full operating system and applications. While the latter is more believable and can capture more detailed information about the attacker, they are more expensive to deploy and maintain. In [7, 6] the authors propose a game-theoretic model that studies various camouflaging signals that honeypots can send to the attacker in order to minimize the chance of being detected. Honeypots were mainly used by the researchers ([35]) who aimed to learn about the attackers: why do they attack, what are their targets and what tools do they use. Probably the most extensive concentration of these researchers is in non-profit organization Honeynet Project[2], founded in 1999. They provide various low and high interactive honeypots for their members and publish gathered information and experience. Honeynet Project demonstrated great gathering capabilities in their series of papers *Know Your Enemy*[3].

Recently, companies began paying attention to honeypot as well. E.g., company Thinkst provides commercial product honeypot hardware equipment, called *Canary*, for intrusion detection system purposes. They contain various operating systems with user configurable open ports. While they are easy to deploy, deciding where to optimally allocate expensive equipment in the network for the best tradeoff between cost and probability of detecting the attacker brings a challenging decision for the administrator, which is not provided. My research focuses on answering such questions.

2.2 Attack Graphs

There are two types of attack graphs common in the literature, (i) *state enumeration attack graphs* and (ii) *dependency attack graphs* [31]. In state enumeration attack graphs vertices represent a *complete* network state and transitions are attacker actions that modify this state. For a given set of propositions that describe a network this representation results in an exponential number of possible states, which limits this representation to small networks [23]. Dependency attack graphs use a more compact representation based on logical statements about the state of the network and actions that modify these statements. We use a special case of dependency attack graph, an AND/OR attack graphs with action uncertainty. Actions have *preconditions* (dependencies), *effects*, *success probabilities*, and *costs* for the attacker [31]. We also use the common monotonicity assumption [2, 24, 20] that once a fact becomes true during an attack it can never become false again as an effect of any action (i.e., there are no negative action effects). Penetration testing companies, such as Core Security, use attack graphs to find the shortest attack path in terms of time for the penetration tester to maximize number of tests in a limited time. More recently,

[2] http://www.honeynet.org
[3] available at http://honeynets.org/papers

in [12] authors combine attack graph with intrusion alerts, active responses and dependency graph for purposes of forecasting future attacks.

The most common analysis of attack graphs includes identifying the optimal subset of vulnerabilities that need to be removed to prevent all known attacks [33] or finding a minimal set of sensors to be placed to the network to detect all known attacks [21]. Some research focuses on computing network-wide security measures and using them to choose countermeasures that result in the highest network security [22, 15]. Typically, these metrics evaluating security countermeasures are based solely on the probability of a successful attack on the network. Computing this probability has several disadvantages: (1) it disregards the possible costs of different attacks (for both the attacker and defender), and (2) it does not provide any information about the order in which the attacker executes individual actions. As a result, the defender can hardly evaluate a real effectiveness of the security measures that depend on the order of attack actions, such as placing honeypots or other sensors in the network. Our approach differs significantly in that we compute optimal contingent attack policies minimizing the expected cost for the attacker, rather than just the probability of successful attack. Contingent attack policies are crucial for our game-theoretic approach as they provide enough information to determine the attacker's expected utility in the game. We also allow for general attack graphs without structural restrictions, even though a simplified variant of attack graphs is shown to be NP-hard in [13].

2.3 Game Theory

Game theory is well-suited to model adversarial reasoning required for the security resource allocation. There have been many applications of game theoretic algorithms deployed in practice [37], e.g., randomized schedules of checkpoints on roadways to Los Angeles International Airport [17], patrolling in US Coast Guards [34], etc. An extensive survey of application of game theory to network security is presented in [29]. Only a few game theoretic models in network security focus on intrusion detection. Most of them focus on removing vulnerabilities or direct counter-actions of the defender after an attack is detected. In [1] authors present a simple and very abstract model of attacker choosing a suitable attack action and defender selecting an intrusion detection strategy. They investigate mainly how the optimal strategies of the players are influenced by availability of different kinds of information players have about their opponents. Schmidt et al. in [32] similarly to Vanek et al. [38] present game theoretic models of placing intrusion detection sensors to specific positions in the topology of the network. While the first work evaluates their methods in a realistic simulation environment, the latter work focusses on the algorithmic aspects and proposes efficient approximate solution techniques scalable to larger networks.

Attack graphs used in combination with the game-theoretic approach for attack intention recognition ([19]), where defender actively monitors the network's IDS alerts as attack progresses to optimally decide what defense actions to perform. In [18] are recent game-theoretic models that use attack graphs, but

these models had unrealistic assumption that the attacker has perfect information about the original network structure. The attackers imperfect information was proposed in [26], however, they use very abstract one step attack action which do not allow the rich analysis of the impact of defender's actions on attacker's decision making process.

3 Contribution

In this section we give high-level description of the research carried out up to now. So far we focused on administrator's actions of allocating honeypots in the network. More detailed contribution is attached to this proposal in form of accepted conference papers in Appendix A.

3.1 Optimal Attack Planning

In analysis of attack graphs, it is often of interest to identify the optimal strategy of the attacker (i.e., which actions to execute in what situation) and its expected cost. For example, comparing the expected cost of the attack to the expected reward of successfully compromising the target indicates if a rational attacker would attack the system at all [5]. In penetration testing, following the optimal attack strategy can save a lot of valuable time [30]. Computing the optimal strategy for the attacker is also a building block in solving various game-theoretic models of interaction between the attacker and defender of a system. In [13] it was shown that a simplified version of this problem is NP-hard.

We use MulVAL [3, 25] software to automatically generate attack graphs. An attack graph consists of atomic attack actions that an attacker can perform. Each action has a defined cost that the attacker pays in order to attempt the action. Additionally, each action has a certain probability of being performed successfully. In [9] (paper in Appendix A) we presented an algorithm to find the optimal attack policy for a specific attack graph. The algorithm iteratively constructs finite horizon Markov Decision Processes (MDP) and uses value iteration approach to compute the optimal attack policy. We introduce several speed-up and pruning techniques to increase the algorithm's scalability: (i) the MDP is not fully constructed, but only a part that is needed to a provide provably optimal attack policy; (ii) we extend Sibling-Class Theorem (Theorem 20 in [13]), defined for attack trees into cyclic attack graphs, which effectively prunes out suboptimal policies; (iii) we use heuristic and branch and bound method to additionally prune out suboptimal policies; and (iv) we use caching to avoid same effort twice. This algorithm solves single-goaled attack graphs with highly motivated attackers. We use it as a building block for further game-theoretic algorithms.

3.2 Perfect-Information Game

In [11] (paper in Appendix A) we introduce a *Stackelberg* (perfect-information) game. In a Stackelberg game we assume that the first player (the defender) commits to a publicly known pure strategy. The second player (the attacker) plays pure best response to the (observed) strategy of the defender. The set of actions for the defender corresponds to deploying one or more honeypots in the network. We assume there is a limited set of possible options and we restrict the defender to use only honeypots that are an exact copy of an existing host on the network in terms of configuration and connectivity (i.e., defender is duplicating a host already in the network). This has several advantages. First, copying an existing machine and obfuscating the useful data on the machine can be done almost automatically with lower cost than configuring a brand new host. Second, it can create much more believable honeypots that will make it hard for the attacker to distinguish them from production systems. Third, if the attacker uses some unknown exploit on the honeypot, we know that the same exploit is available on one of the production machines. After the defender deploys honeypots into the network, the attacker selects the optimal attack policy (in terms of expected reward) to compromise the network, taking into account the network structure and the defender's honeypot allocation.

In this paper we present a comprehensive analysis of the proposed algorithm on theoretical level as well as experimentally on several networks. We are aware of the fact that in reality the attack graph, available for the defender, only estimates the attacker's real behavior. Therefore, we also provide results for defender's regret for modeling δ-imprecise attack graph, namely, attack action costs, attack action success probabilities and rewards for the host types.

3.3 Imperfect-Information Game

Until now we assumed that it is known to the attacker what network he attacks and what honeypots the defender deployed. 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. In our paper [10] (paper in Appendix A) we introduce imperfect information game, where 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.

To capture the set of networks we model the game as a three-stage extensive-form game with a specific structure. First, *Nature* selects a network from the set of possible networks with the probabilities corresponding to the prior attackers beliefs about the likelihood of the different networks. Second, 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, resulting in attacker's information sets. Third, 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.

Solving this game (meaning finding its Stackelberg Equilibrium) introduced additional computational challenges that must have been solved. For instance, in order to compute the attacker's optimal attack policy we had extend previous MDP algorithm to a Partially Observable MDP (POMDP) with finite horizon model. More importantly, finding Stackelberg Equilibrium as a solution concept in games with information sets cannot be done using backward induction, approach used in the previous game model and current state-of-the-art algorithms have very low scalability. We introduced three algorithms that find suboptimal quality strategies for the defender in reasonable time and experimentally compare them. A particularly interesting algorithm approximates Stackelberg equilibrium by computing the commitment to correlated equilibrium solution concept. This algorithm lead to developing new iterative algorithm to find Strong Stackelberg Equilibrium in general-sum games which outperforms the best previously known algorithm [4] by several orders of magnitude, published in [39] (included in Appendix A).

3.4 List of Publications

In this section I present list of relevant publications in chronological order. All of them can be found in Appendix A (except the last one, which is still under review).

1. **Computing Optimal Policies for Attack Graphs with Action Failures and Costs**. Durkota, K., Lisý, V.: In proceedings of 7th European Starting AI Researcher Symposium (STAIRS) at European Conference on Artificial Intelligence (ECAI), pages 101–110, 2014.

2. **Game-Theoretic Algorithms for Optimal Network Security Hardening Using Attack Graphs**. Durkota, K., Lisý, V., Kiekintveld, C., Bošanský, B.: In proceedings of International Conference On Autonomous Agents & Multiagent Systems (AAMAS) 2015. (Microsoft rank: 82).

3. **Optimal Network Security Hardening Using Attack Graph Games**. Durkota, K., Lisý, V., Bošanský, B., Kiekintveld, C. In proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pp. 7–14 (2015) (Microsoft rank: 130).

4. **Approximate Solutions for Attack Graph Games with Imperfect Information**.Durkota, K., Lisý, V., Bošanský, B., Kiekintveld, C.: In proceedings of Decision and Game Theory for Security (GameSec), pages 228–249. Springer, 2015.

5. **Using Correlated Strategies for Computing Stackelberg Equilibria in Extensive-Form Games**. Čermák, J., Bošanský, B., Durkota, K., Lisý, V., Kiekintveld, C. In proceedings of Association for the Advancement of Artificial Intelligence (AAAI) 2016. (Microsoft rank: 132).

6. **Solving Honeypot Allocation Problem using Game Theory and Attack Graph**. IEEE Intelligent Systems (special issue). 2016 (Under Review).

4 Dissertation Plan

The ongoing work greatly supports the dissertation objectives. There are three main directions of extending current game models.

4.1 Extending Network Hardening Actions

So far we payed attention only to the administrator's actions for allocating honeypots in his network. Honeypots plays somehow against the concept of the Stackelberg Equilibrium in assumption that the defender openly announces his strategy to the attacker. In reality the defender does not want the attacker to know where are the honeypots.

In this part I would like to focus on the defender's greater repertoire of possible actions to harden his network. Goal is to generalize our game model to capture other network hardening actions. The general game-theoretic framework seems to be limited to those actions that have influence on the attacker's attack graph, i.e., firewall rules, vulnerability patches, service reduction, etc. Actions that have no impact on the attack graph, therefore on the attacker's behaviour, are inappropriate for our model, as their contribution cannot be evaluated. This part will expand the problem range (**Objective 1**) to capture additional defender's actions. This research package will be proceeded in the following steps.

Hardening Actions First, I will research for other network hardening action that are available for the administrator. Each action will be thoroughly analyzed to determine its availability, application cost, impact on the attacker's behaviour (attack graph), etc.

Theoretical Analysis Each action will be implemented into the existing framework. The resulting algorithm will be analyzed in terms of scalability, computed strategy quality, etc.

4.2 Hardening Web Security using Honeypots

As a result of transfer most of the computer softwares and user services to the web applications and the fact that networks are closely protected by well advanced Next Generation Firewalls and Intrusion Detection Systems, the web applications become the most targeted and attacked platform. Moreover, lack of available computer networks that can be compromised makes it hard to evaluate the honeypots effectiveness in networks. In order to empirically evaluate their effectiveness, I would like to analyze honeypots in more flexible, available and less expensive domain, the websites and web-applications. *Honeytoken* is a similar concept to honeypots, however, they are not constrained to be used as

a service or host, and therefore applicable for web applications. Honeytoken can be a fake file, fake link or any other type of information that misleads the attacker. Honeytokens can be used in websites to misinform and mislead the attacker. A typical example is adding decoy parameter into url, e.g., in `www.domain.com/admin.php?pass=word&access=false` the parameter `access` may be fake one. If attacker attempts to change its value to `true`, he is identified and some countermeasures can be applied, e.g., his connection is redirected to a *sandbox*, the identical website with fake content. Another typical example is adding a decoy files, e.g., www.domain.com/admin.php, for the same purpose. Additionally, the website may contain links that are invisible for the human but automatic attacker's may follow them. These techniques enable the websites to divert or mitigate the detected attacks.

Goal is to develop a game-theoretic framework to guide the web administrator in decision making of what type, number and targets of honeytokens to deploy in web application with a limited budget. I intend to compute the optimal strategy for the defender against rational attacker and analyze the quality of the found solution. In this part I will expand the range of the problem (**Objective 1**) to web security and carry out extensive empirical evaluation (**Objective 4**). The plan to address this problem by the following statement of work:

Web administrator's Actions First, it is necessary to identify possible honeytokens for the defender to deploy into a web application for the model. Primarily I will consider actions proposed in research publications as well as few own ideas that are suitable for this problem.

Web Attack Graphs I will analyze the attacker's possible actions of compromising the web application based on automatic scanning tools for vulnerabilities.

Models and Algorithms Finally, a principal novel contribution I propose a game-theoretical model of the defender/attacker interactions and provide a model of their utility functions. I will propose several solution concepts of the suggested game model and provide an algorithm for solving the proposed game by finding an optimal and stable strategy for each player.

4.3 Improvement Malware Detection using Honeypots (Optional)

This part of my dissertation work I would like to dedicate to a particularly interesting security problems, such as Botnet and/or Advanced Persistent Threat (APT) problems. Botnet is a collection of infected computers in the network that are controlled by an adversary, called botmaster, who communicates with botnets by broadcasting commands that the he wants them to perform. Botnets are often used for DoS attack, adwares, e-mail spams, click frauds, etc. APT, on the other hand, is done by very motivated attackers for stealth of the valuable data of the targeted organization, such as national defense, manufacturing or financial industry. Both, botnets and APT, are difficult to detect as they are subtle and do not disrupt the network (in contrast to DOS attacks, etc.).

I believe that honeypots could be used to improve their detection. By allocating vulnerable honeypot into the network, it is likely that the honeypots will be infected if any host in the same subnetwork is already infected. In case of botnet detection, the honeypots will become one of the botmaster's arsenal host that he remotely controls. Having botnet infected, gives variety of possibilities for the administrator how to use it for own advantage. E.g., if all bots receive the same command from the botmaster, e.g., to begin a DoS attack, they are likely to produce the same or similar traffic. At this point, the administrator detect infected hosts based on correlated traffic that they produce with the traffic of the infected honeypot. Botnets can be modeled using botnet attack graphs (or other botnet behaviour model). This opens new game-theoretic research opportunities for honeypot usage in botnet detection. E.g., where it is best to allocate honeypots in the network to maximize the probability that at least one honeypot will get infected if there is an infected host. On the other hand, the attacker decides which hosts should he infect to avoid infecting honeypots, or what attack actions to send to the bots to avoid correlated behaviour. Botnets can be modeled using botnet attack graphs (or other botnet behaviour model), and actions between the bots and can modeled. Similar research steps could be applied for APT detection, where attacker are real hackers. This part is dedicated to fulfill the (**Objective 4**) with empirical evaluation of our approach to a specific scenario.

Honeypot Allocation Continuing on previous research package, we can further analyze game-theoretically where it is best to allocate honeypots in the network so that: (i) it is likely to get infected if any other host in the network is infected, (ii) it is likely to produce correlated traffic with other infected hosts, and (iii) the cost of allocating the honeypots is minimal.

We know that the botnets, reps. APT attackers, may become cautious in attacking the neighbour hosts because of the existence of the honeypots. The defender's decisions (where to allocate honeypot) and attacker's decisions (which hosts to infect) can be analyzed using game theory. We will further investigate the game-theoretic setting and develop algorithm to support and help the defender with the security decision making.

4.4 Deployment (Optional)

After the models are theoretically researched deep enough, the models' applicability will be evaluated in the real-world. Evaluation of the administrator's different network security hardening actions can be evaluated using collected data from cyber-defense exercises, such as data from Baltic Cyber Shield Cyber Defense Exercise, provided by FOI (Swedish Defense Research Agency)[4] ; or a virtual network can be created using KYPO [8] platform, which serves for cyber defense exercise, and run our own experiments. Additionally, I will use honeypots in more focused scenarios, such as for botnet/APT detection, in a real world experiments. It will be approached through collaborative research

[4]available at ftp://download.iwlab.foi.se/dataset/

with companies, such as Cisco, for evaluation of our methods against (or in combination with) the methods that companies apply nowadays. The quantitative results can be very valuable to the security and game-theoretical research communities. This research part is conditioned by the collaboration with other parties and quality of the accessible data.

5 Conclusion

The current state of progress on the dissertation shows, that the dissertation objectives were rightly chosen an are being fulfilled. The network security is still an opened and important problem. Network security, games theory and their combination as presented in this work are hot topics in the research communities. This dissertation uses game theory, well supported theoretically and experimentally, to show significant progresses of the state of the art in security domain. Even more novel techniques are to be proposed, formalized and evaluated.

Appendices

A Conference Papers

In this section I present relevant conference papers that have been accepted and published, as listed in Section 3.4.

Computing Optimal Policies for Attack Graphs with Action Failures and Costs

Karel DURKOTA and Viliam LISY
Agent Technology Center, Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague
{karel.durkota, viliam.lisy}@agents.fel.cvut.cz

Abstract. An attack graph represents all known sequences of actions that compromise a system in form of an and-or graph. We assume that each action in the attack graph has a specified cost and probability of success and propose an algorithm for computing an action selection policy minimizing the expected cost of performing an attack. We model the problem as a finite horizon MDP and use forward search with transposition tables and various pruning techniques based on the structure of the attack graph. We experimentally compare the proposed algorithm to a generic MDP solver and a solver transforming the problem to an Unconstrained Influence Diagram showing a substantial runtime improvement.

Keywords. optimal policy, attack graph, markov decision process, and-or graph

Introduction

Attack graphs (AG) are a popular tool for analysing and improving security of computer networks, but they can be used in any domain, where attacks consist of multiple interdependent attack actions. Attack graphs capture all the known sequences of actions that may lead to compromising a system, and they can contain additional information, such as the cost of individual actions and the probability that the actions will be successfully executed. AGs can be used to evaluate risks and design appropriate countermeasures.

In analysis of attack graphs, it is often of interest to identify the optimal strategy of the attacker (i.e., which actions to execute in what situation) and its expected cost. For example, comparing the expected cost of the attack to the expected reward of successfully compromising the target indicates if a rational attacker would attack the system at all [3]. In penetration testing, following the optimal attack strategy can save a lot of valuable time [7]. Computing the optimal strategy for the attacker is also a building block in solving various game-theoretic models of interaction between the attacker and defender of a system. Furthermore, a problem of computing the optimal attack strategy can also be seen as a complex variant of the generic problem of probabilistic and-or tree resolution analysed in AI research [4].

In this paper, we propose an algorithm for computing the optimal attack strategy for an attack graph with action costs and failure probabilities. Unlike previous works assuming that the attack graph is a tree (e.g, [7]) and/or computing only a bound on the actual value (e.g., [3]), we compute the exact optimum and we do not impose any

restriction on the structure of the attack graph. Specifically, our approach allows the attack graph to contain (even oriented) cycles and to have actions with probabilities and costs as inner nodes of the attack graph.

The drawback of our approach is that even a simplified variant of this problem has been shown to be NP-hard in [4]. As a result, we solve it by a highly optimized search algorithm and experimentally evaluate its scalability limitations. We show that the problem can be mapped to solving a finite horizon Markov decision process (MDP) and how the information about the structure of the attack graph can be used do substantially prune the search space in solving the MDP. We compare the proposed approach to recently published method for solving this problem [6] and to a recent version of a generic MDP solver from the International Planning Competition 2011 [5], showing that the proposed method scales orders of magnitude better.

1. Background and Definition

1.1. Attack Graph

AG is a directed graph consisting of two types of nodes: (i) *fact nodes*, that represent facts that can be either true or false, and (ii) *action nodes*, that represent actions that the attacker can perform. Each action has *preconditions* – a set of facts that must be true before action is performed and *effects* – a set of facts that becomes true if action is successfully performed. Moreover, every action has associated probability $p \in (0,1]$ – which denotes the probability that action succeeds and its effects become true, and with probability $1-p$ action fails and attacker cannot repeat this action anymore. We assume that attacker cannot repeat actions for couple of reasons: (i) if actions are correlated and have static dependencies (installed software version, open port, etc.), another attempts to use the same action would result alike, and (ii) if we allow infinitely many repetitions, optimal attack policy (explained further) would collapse into a linear plan with attempting for each action until action succeeds[3]. Finally, each action has associated cost c; if attacker decides to perform action a, he will pay the cost c, regardless whether the action is successful or not.

Definition Let Attack Graph be a 5-tuple $AG = \langle \mathrm{F}, \mathrm{A}, \mathrm{g}, p, \mathrm{c} \rangle$, where:

- F is a finite set of facts
- A is a finite set of actions, where action $a : pre \rightarrow eff$, where $pre \subseteq \mathrm{F}$ is called *preconditions* (we refer to them as $pre(a)$) and $eff \subseteq \mathrm{F}$ is called *effects we refer to them as* $eff(a)$
- $\mathrm{g} \in \mathrm{F}$ is the goal
- $p : \mathrm{A} \rightarrow (0,1]$ is the probability of action to succeed (we use notation p_a for probability of action a to succeed, and with $p_{\bar{a}} = 1 - p_a$ the probability of action to fail)
- $\mathrm{c} : \mathrm{A} \rightarrow \mathbb{R}^+$ is cost of the action (we use notation c_a for cost of the action a).

We use the following terminology: we say that fact f *depends* on action a if $f \in eff(a)$, and similarly, action a depends on f if $f \in pre(a)$

Example of such Attack Graph is in Fig. 1. Diamonds are the inner fact-nodes, that are initially false, but can be activated performing any action on which the facts depend

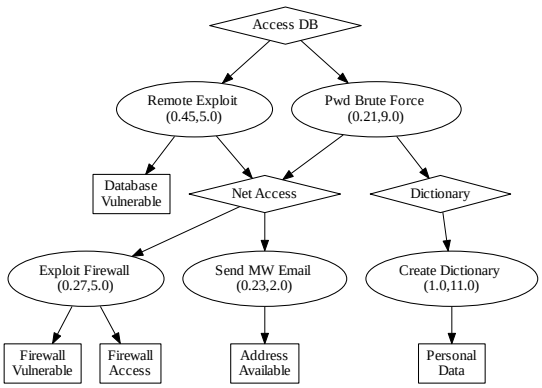


Figure 1. Simple attack graph which shows possible ways how to achieve access to the database (fact node "`Access DB`"). Diamonds are the inner fact-nodes (initially false) that can be turned true, while rectangles are the leaf fact nodes, which are always true. Ellipses depict actions that attacker can perform with success probability p and cost c.

upon. Rectangles represent leaf fact-nodes, that are initially true. Ellipses are the actions that attacker can perform with probability of success p and cost c. In our example we represent action with its name and the couple (p,c). Attacker's goal is to activate fact "`Access DB`" (obtain an access to DB).

The probabilities and costs of the actions can be obtained using Common Vulnerability Scoring System (CVSS)[1] from, i.e., National Vulnerability Database, which scores different properties of vulnerabilities. Probabilities could be computed for example from the access complexities, exploitabilities or availability impacts of the vulnerabilities, whereas costs could be computed from number of required authentication in order to a vulnerability, etc.

1.2. Attack Policy

Solving the AG means to find a policy, that describes what action should attacker perform in every possible evolution of the attack procedure. Fig. 2 depicts optimal policy ξ_{opt} for the problem from Fig. 1, where attacker first attempts to perform action "`Send MW Email`"; if action is successful, he follows the right (sub)policy (solid arc), thus performing action "`Remote Exploit`", otherwise the left (sub)policy (dashed arc), thus action "`Exploit Firewall`", and so on.

Definition An *attack policy* is an oriented binary tree ξ for evaluating attack graph AG. Nodes of ξ are actions, arcs are labeled $+$ or solid line (if parent action was successful) and $-$ or dashed line (if parent action was unsuccessful), and whose leaf-nodes are either $\boxplus$ resp. $\boxminus$ representing successful reps. unsuccessful attack.

[1] www.first.org/cvss

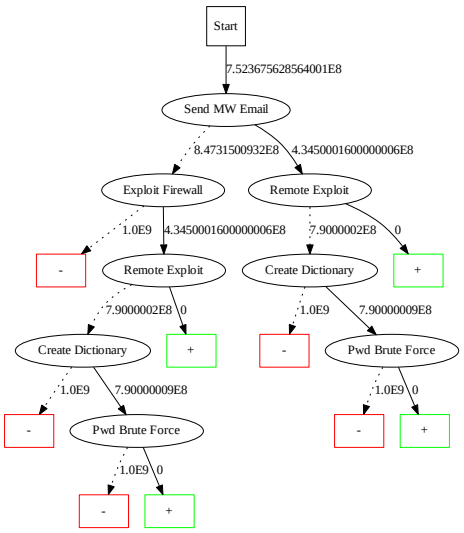


Figure 2. Optimal policy for a simple attack graph from Fig. 1. Attacker should follow solid arcs if previous action was successful, otherwise follow dashed line. Values at arcs represent the expected costs for attacker.

Definition The *expected cost* of an attack policy ξ is a cost over all possible evolutions of the policy. Let ϕ_χ be (sub)tree of ξ rooted at a node χ labeled with an action a, then expected cost of ϕ_χ can be computed recursively as follows:

$$\mathscr{E}(\phi_\chi) = c_a + p_a \times \mathscr{E}(\phi_{\chi^+}) + p_{\bar{a}} \times \mathscr{E}(\phi_{\chi^-})$$

where ϕ_{χ^+} (ϕ_{χ^-}) is the subtree rooted at χ's $+$ branch ($-$ branch) and in the leaf-nodes of the policy is a penalty if attack is unsuccessful $\mathscr{E}(\boxminus) = penalty$ and reward if successful $\mathscr{E}(\boxplus) = reward$.

When we decide which of the two policies, either ϕ_χ rooted at χ labeled with an action a or ϕ_ψ rooted at ψ labeled with action b have lower expected cost, we assume that after performing action a, resp. b, attacker follows an optimal policy. In this case, we override our notation of $\mathscr{E}(\phi_\chi)$ resp. $\mathscr{E}(\phi_\psi)$ to simply $\mathscr{E}(a)$ resp. $\mathscr{E}(b)$.

We assume that our attacker is a *motivated* attacker, that is they continue in attack as long as there are actions that may lead to the goal, regardless of the cost of the attacks. Motivated attacker ceases the attack only when there is no sequence of actions that could result in achieving the goal. Having assumed this type of attacker and the fact that attacks are *monotonic*, meaning that consequence of attack preserves once is achieved [1] (once the fact becomes true, it cannot become false again), it can be shown that every policy, regardless on the order of the action, will have equally the same probability of achieving the goal. Note, that the expected cost of the policy consist of two parts: the probability of achieving the reward or the penalty and the expected cost of the action costs. The probability of successful attack is always the same, thus every policy will have the same

expected cost of the penalty/reward. Thus, it essentially makes no difference whether we choose to reward the attacker for successful attack or penalize for an unsuccessful attack. In fact, distinct policies have different expected costs only because of the different action ordering which imposes different sequences of their costs.

Definition A policy ξ_{opt} is *optimal* if it has the minimal expected cost among all possible policies, thus $\forall \xi \in \Xi : \mathscr{E}(\xi_{opt}) \leq \mathscr{E}(\xi)$, where Ξ is a set of all policies.

Proposition 1.1 *In the optimal policy ξ_{opt} for every (sub)policy ϕ_χ rooted at a node χ labeled with an action a following is true: $\mathscr{E}(\phi_\chi) \leq \mathscr{E}(\phi_{\chi^-})$.*

We will prove it by contradiction. Assume that $\mathscr{E}(\phi_\chi) > \mathscr{E}(\phi_{\chi^-})$ is true. Then due to the monotonicity property the attacker could have followed the (sub)policy ϕ_{χ^-} even before performing action a, which would have saved him the cost of the action c_a. But then this new policy would have had lower expected cost than the policy ϕ_χ, which violates our assumption that ϕ_χ is an optimal policy.

Proposition 1.2 *In the optimal policy ξ_{opt} for every (sub)policy ϕ_χ rooted at a node χ labeled with an action a following is true: $\mathscr{E}(\phi_{\chi^-}) \geq \mathscr{E}(\phi_{\chi^+})$.*

$$\mathscr{E}(\phi_\chi) = c_a + p_a * \mathscr{E}(\phi_{\chi^+}) + p_{\bar{a}} * \mathscr{E}(\phi_{\chi^-}) \tag{1}$$

$$\mathscr{E}(\phi_{\chi^-}) \geq c_a + p_a * \mathscr{E}(\phi_{\chi^+}) + p_{\bar{a}} * \mathscr{E}(\phi_{\chi^-}) \tag{2}$$

$$\mathscr{E}(\phi_{\chi^-}) \geq c_a + p_a * \mathscr{E}(\phi_{\chi^+}) + c_a/p_a \tag{3}$$

1.3. Markov Decision Process

We solve this problem by modeling it as Markov Decision Processes (MDP) [2] which is defined as 4-tuple $\langle S, A, P_\cdot(\cdot,\cdot), R_\cdot(\cdot,\cdot)\rangle$, where:

- S is a finite set states, in our case state is a set of performed actions and label whether the action a was successful (a) or not ($\bar{a}$);
- A is a set of actions, which is equal to the set of actions in the attack graph
- $P_a(s,s')$ is a probability that action a, performed in state s, will lead to state s'; in our case, if action a, with probability p_a is successful, then state $s' = s \cup \{a\}$; if action a is unsuccessful, then state $s' = s \cup \{\bar{a}\}$
- $C_a(s,s')$ is an immediate cost payed after transition to state s' from state s; in our case $C_a(s,s') = c_a$ in all transitions, except when s' is a terminal state, then $C_a(s,s') = c_a - reward$ if goal is achieved in s' and $C_a(s,s') = c_a + penalty$ if goal is not achieved in s'.

Optimal solution is such a policy of MDP, that minimizes an overall expected cost.

2. Algorithm

2.1. Basic Approach

Basic approach is to use MDP with finite horizon, e.g. exhaust every possible action at every decision point and select action having minimal expected cost. In fact,

we use this approach with several pruning techniques which speed up this computation. In Fig. 3 is an example of MDP search of our running example from Fig 1. The root of the MDP is a decision point where we need to decide which of the action among `"Exploit Firewall"`, `"Send MW Email"` and `"Create Dictionary"` is the best to perform. In a naive approach we explore every possible scenario and compute their expected costs $\mathscr{E}(\texttt{"Exploit Firewall"}), \mathscr{E}(\texttt{"SendMW Email"})$ and $\mathscr{E}(\texttt{"Create Dictionary"})$. We choose the action with the minimal expected cost.

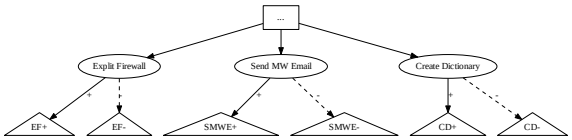


Figure 3. In naive approach we explore every possibility and select action having minimal expected cost.

For performance enhancement, we make use of *transposition tables*, that is: we cache states for which we have computed expected cost and an optimal (sub)policy and reuse these results in future, should we encounter the same state again.

2.2. Sibling-Class Theorem

In [4] authors deal with "probabilistic and-or tree resolution" (PAOTR) problem, mainly for and-or trees with independent tests without preconditions, for which they constructed and proved the Sibling-Class Theorem. Independently, authors in [3] show the same theorem. The Sibling-Class Theorem states, that the leaf-node actions can be grouped into the sibling-classes within which actions' ordering can be determined by simply sorting their R-ratios; hence, no state-search exploration is necessary within sibling-class, only between the sibling classes. Two actions belong to the same sibling class if they have common parent in the and-or tree. As they consider inner nodes to be either AND or OR node, naturally there are two types of sibling classes: the AND-sibling classes and the OR-sibling classes. R-ratios of an action is computed as follows:

$$R(a) = \frac{p_a}{c_a} \text{ if action } a \text{ is in OR-sibling class} \tag{4}$$

$$R(a) = \frac{p_{\bar{a}}}{c_a} \text{ if action } a \text{ is in AND-sibling class} \tag{5}$$

Conjecture 2.1 *Sibling-Class Theorem for and-or trees without preconditions can be applied to an and-or graph with precondition using following rules for creating OR-Sibling Classes:*

- *actions a and b belong to the same OR-Sibling class* iff*:* $pre(a) = pre(b) \wedge |pre(a)| = |pre(b)| = 1$.

and following rules for AND-Sibling Class:

- *action a and b belong to the same AND-Sibling class* iff*:* $pre(a) \neq pre(b) \wedge |pre(a)| = |pre(b)| = 1 \wedge (\exists c \in A : pre(a) \in \mathit{eff}(c) \wedge pre(b) \in \mathit{eff}(c))$.

Action $a \in A$ cannot be pruned iff*:* $|pre(a)| > 1 \vee (\exists c_1, c_2 \in A : c_1 \neq c_2 \wedge pre(a) \in \mathit{eff}(c_1) \wedge pre(a) \in \mathit{eff}(c_2))$.

The Sibling Theorem is proved only for and/or trees, while we empirically checked and use it for and/or graphs.

Example Assume we have the same problem as in Fig. 3 and we come to the same decision point as previously. But now we computed $R(\texttt{"Exploit Firewall"}) = \frac{0.27}{5.0} = 0.054$, $R(\texttt{"Send MW Email"}) = \frac{0.23}{2.0} = 0.115$ and $R(\texttt{"Create Dictionary"}) = \frac{1.0}{11.0} = 0.091$ and we know that actions `"Exploit Firewall"` and `"Send MW Email"` have the same parent node `"Net Access"`, thus belong to the same OR-sibling class, while action `"Create Dictionary"` belongs to a separate sibling class. Now we explore only actions that have maximum R-ratios in each sibling class, thus only actions `"Send MW Email"` and `"Create Dictionary"`, and action `"Exploit Firewall"` surely will not be the first action in the optimal policy. Fig. 2.3 depicts nodes that must be explored (white nodes), and nodes that are pruned (grey nodes).

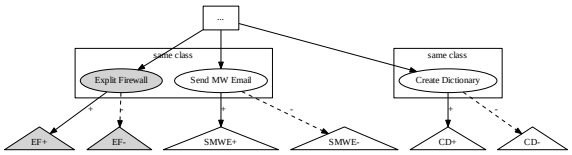


Figure 4. This figure presents nodes that are explored (white) and nodes that can be pruned (grey) in the MDP search if we know that actions `"Exploit Firewall"` and `"Send MW Email"` are in the same sibling class and action R-ratios.

2.3. Branch and Bounds

As another pruning technique we use branch and bounds. For this technique we reuse previously computed expected costs of the subtrees of the MDP to prune future subtrees if know that an optimal solution cannot exist there. Specifically, when we face the decision either utilize (sub)policy ϕ_a – starting with an action a – or (sub)policy ϕ_b – starting with action b – we choose policy ϕ_b only if $\mathscr{E}(\phi_b) < \mathscr{E}(\phi_a)$, implying:

$$\mathscr{E}(\phi_b) < \mathscr{E}(\phi_a) \tag{6}$$

$$c_b + p_b * \mathscr{E}(\phi_{b^+}) + p_{\bar{b}} * \mathscr{E}(\phi_{b^-}) < \mathscr{E}(\phi_a) \tag{7}$$

$$c_b + p_b * \mathscr{E}(\phi_{b^+}) + p_{\bar{b}} * \mathscr{E}(\phi_{b^+}) < \mathscr{E}(\phi_a) \tag{8}$$

$$c_b + \mathscr{E}(\phi_{b^+}) < \mathscr{E}(\phi_a) \tag{9}$$

$$\mathscr{E}(\phi_{b^+}) < \mathscr{E}(\phi_a) - c_b \tag{10}$$

where from (8) to (9) we used property of the optimal policy that $\mathscr{E}(\phi_{b^+}) \leq \mathscr{E}(\phi_{b^-})$, and then fact that $p_b + p_{\bar{b}} = 1$. Thus, $\mathscr{E}(\phi_a) - c_b$ is an upper-bounds for $\mathscr{E}(\phi_{b^+})$. If anytime

during the computation it exceeds this bound, we can immediately stop the computation of the b^+ branch.

Similarly, having computed the $\mathscr{E}(\phi_b)$, we can bound again the branch $\mathscr{E}(\phi_{b^-})$ as follows

$$\mathscr{E}(\phi_b) < \mathscr{E}(\phi_a) \tag{11}$$

$$c_b + p_b * \mathscr{E}(\phi_{b^+}) + p_{\bar{b}} * \mathscr{E}(\phi_{b^-}) < \mathscr{E}(\phi_a) \tag{12}$$

$$p_{\bar{b}} * \mathscr{E}(\phi_{b^-}) < \mathscr{E}(\phi_a) - c_b - p_b * \mathscr{E}(\phi_b) \tag{13}$$

$$\mathscr{E}(\phi_{b^-}) < (\mathscr{E}(\phi_a) - c_b - p_b * \mathscr{E}(\phi_b))/p_{\bar{b}} \tag{14}$$

Example Assume different example, where we face the problem of choosing the best (sub)policy ϕ_a, ϕ_b, ϕ_c and ϕ_d. Branches $\mathscr{E}(\phi_{a^+})$ and $\mathscr{E}(\phi_{a^-})$ we must compute to obtain $\mathscr{E}(\phi_a)$. Next, we compute expected cost $\mathscr{E}(\phi_{b^+})$ and assume that it turns out to be higher than $\mathscr{E}(\phi_a)$, hence, we can prune the computation of the branch b^-, as action b will never have less expected cost then action a. Next, let's say $\mathscr{E}(\phi_{c^+})$ in the branch c^+ turned out to be lower than $\mathscr{E}(\phi_a)$. It means that we can upper bound the $\mathscr{E}(\phi_{c^-})$ by $\frac{\mathscr{E}(\phi_a)-c_c-p_c*\mathscr{E}(\phi_c)}{p_{\bar{c}}}$. Let's say that $\mathscr{E}(\phi_{c^-})$ obeyed the bound, thus, action c is the best action that attacker can perform so far. Note, that it is unnecessary to compute total expected cost of $\mathscr{E}(\phi_c)$ to determine that it is lower then $\mathscr{E}(\phi_a)$, it is direct implication from the fact that it satisfied both bound conditions. Finally, during the computation of branch d^+ it turned out to violate new upper bound $\mathscr{E}(\phi_c) - c_d$, thus its computation was terminated and branch d^- was pruned as well.

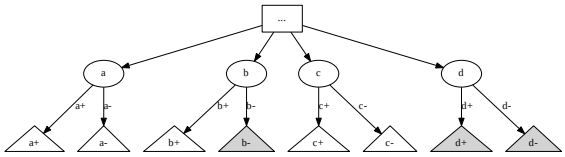


Figure 5. This figure presents nodes that are explored (white) and nodes that can be pruned (grey) due to the branch and bound technique.

2.4. Heuristics

Finally, we designed heuristic approach to compute the lower bound of the expected cost of an attack graph by setting costs of the actions to zero and taking into account only the actions' probabilities. This relaxation gives us freedom in action ordering as any (valid) ordering will produce exactly the same probability of success of the policy and thus the overall expected cost. We use this heuristics in two ways: (i) if computed heuristics exceeds the given upper bound, this branch of computation can be pruned, and (ii) according the heuristics we order the action in which we compute remaining actions'

expected costs. If we start with the most promising action, more of the future branches might get pruned.

Nevertheless, we came across an issue in this approach due to the fact that our attack representation is not a tree but a directed cyclic graph. Which means that performing an action can be beneficial in several possible branches at once if action has more then one root-node paths in the attack graph, which results, that the probability of the action will be count multiple times into the overall probability of success is increased. This causes expected cost, computed as $(1 - probability) * penalty$ to decrease. Since we minimize the expected cost this issue keeps the heuristics still admissible.

3. Experiments

We experimentally compared our algorithm with two other approaches, namely Unconstrained Influence Diagrams, and using probabilistic planner from International Planning Competition.

3.1. Guido approach

This approach, described in [6], converts an attack graph into an Unconstrained Influence Diagrams (UID) — a graphical representation of a decision situations using probabilistic interference — upon which existing solvers can be run. We ran a Guido solver as described in the article. This approach showed to be insufficiently scalable for the problems with large (>20 actions) attack graphs.

3.2. Probabilistic planning

As an other approach, we decided to use a domain independent probabilistic planner SPUDD that competed in International Plannign Competition (IPC) in 2011. SPUDD is based on iterative value computation of MDP and uses own specification language. Since it computes MDP, it needs to have set either discount factor $\gamma = [0,1)$, or $\gamma = 1$ and the horizon set to an integer. For our purposes, discount factor γ must be set to 1, hence horizon had to be chosen appropriately. To ensure that SPUDD finds an optimal solution, we chose to set the horizon to number of actions in the attack graph.

3.3. Experiment settings

We experimentally ran and compared our algorithm DynProg, Guido and SPUDD approaches on the three different realNetwork frameworks with different configurations. We ran experiments on Intel 3.5GHz with memory resource up to 10GB. In DynProg we set the $penalty = 10^9$ and $reward = 0$. In Tab. 1 we present running times of each approach.

4. Conclusion and Future Works

Our algorithm showed to outperform other two approaches in time complexity and scalability. Unfortunately, it ofter runs out of the memory due to the transposition tables and

Problem	DynProg [ms]	Guido [ms]	SPUDD [ms]
Local+2	**51**	85	1000
Local+3	**155**	546	11000
Local+4	**443**	76327	70000
Local+5	**5389**	(OoM)	656000
Local+6	(OoM)	(OoM)	**6152000**
Cross2	**4**	408	1000
Cross3	**38**	23796	9000
Cross4	**504**	(OoM)	287000
Cross5	**3587**	(OoM)	8373000
Cross6	**60351**	(OoM)	(OoT)
LocalChain3-3	**0**	9	**0**
LocalChain4-4	**0**	70	1000
LocalChain5-5	**0**	1169	3000
LocalChain6-6	**0**	17133	23000

Table 1. Time comparison of DynProg, Guido and SPUDD approaches over three types of problems with different complexities. Shortcuts: (OoM) - Out of Memory, (OoT) - Out of Time ($> 10^7$ms).

very large search state-space anyway. Other optimizations can be proposed, as better representation of a state or more accurate heuristics for better pruning. This algorithm can be used in game theoretic manner in couple of ways. Here we present two directions: (i) determine what honeypot configurations maximize the probability that an attacker would be detected during their attacks on the realNetwork and (ii) for security hardening determining which subset of vulnerabilities should administrator fix in order to secure the realNetwork, that is, that for the attacker it is not worth to attack to begin with.

Acknowledgement

This research was supported by the Office of Naval Research Global (grant no. N62909-13-1-N256) and Czech Ministry of Interior grant number VG20122014079.

References

[1] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based realnetwork vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224. ACM, 2002.

[2] Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956.

[3] Ahto Buldas and Roman Stepanenko. Upper bounds for adversaries utility in attack trees. In Jens Grossklags and Jean Walrand, editors, *Decision and Game Theory for Security*, volume 7638 of *Lecture Notes in Computer Science*, pages 98–117. Springer Berlin Heidelberg, 2012.

[4] Russell Greiner, Ryan Hayward, Magdalena Jankowska, and Michael Molloy. Finding optimal satisficing strategies for and-or trees. *Artificial Intelligence*, 170(1):19–58, 2006.

[5] Jesse Hoey, Robert St-Aubin, Alan J Hu, and Craig Boutilier. Spudd: Stochastic planning using decision diagrams, 1999.

[6] Viliam Lisý and Radek Píbil. Computing optimal attack strategies using unconstrained influence diagrams. In *Intelligence and Security Informatics*, pages 38–46. Springer, 2013.

[7] Carlos Sarraute, Gerardo Richarte, and Jorge Lucángeli Obes. An algorithm to find optimal attack paths in nondeterministic scenarios. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, AISec '11, pages 71–80, New York, NY, USA, 2011. ACM.

Game-Theoretic Algorithms for Optimal Network Security Hardening Using Attack Graphs

(Extended Abstract)

Karel Durkota, Viliam Lisý
Dept. of Computer Science
FEE, CTU in Prague
{durkota,lisy}@agents.fel.cvut.cz

Christopher Kiekintveld
Dept. of Computer Science
Univ. of Texas at El Paso, USA
cdkiekintveld@utep.edu

Branislav Bošanský
Dept. of Computer Science
Aarhus University, Denmark
bosansky@cs.au.dk

ABSTRACT

In network security hardening a network administrator may need to use limited resources (such as honeypots) to harden a network against possible attacks. Attack graphs are a common formal model used to represent possible attacks. However, most existing works on attack graphs do not consider the reactions of attackers to different defender strategies. We introduce a game-theoretic model of the joint problem where attacker's strategies are represented using attack graphs, and defender's strategies are represented as modifications of the attack graph. The attack graphs we use allow for sequential attack actions with associated costs and probabilities of success/failure. We present an algorithm for an computing attack policy that maximizes attacker's expected reward and empirical results demonstrating our methods on a case study network.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Algorithms, Economics, Performance, Experimentation

Keywords

attack graphs; optimal attack policy; and-or graph; honeypots; game theory; network security

1. INTRODUCTION

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

We focus on network security hardening problems in which a network administrator (defender) reduces the risk of attacks on the network by introducing honeypots (fake hosts or services) as intrusion detection sensors into their network [7]. Deciding how to optimally allocate these resources to reduce the risk of undetected attacks on a network is a challenging decision for the defender. We use game theory to model this adversarial interaction and to determine the best way to use honeypots against a well-informed attacker. We introduce a novel game-theoretic model for network hardening using honeypots extends Stackelberg security games [8] by adopting a compact representation of strategies for attacking computer networks called *attack graphs*.

Attack graphs (AGs) can represent a rich space of attacker actions sequences for compromising a specific computer network. AGs can be automatically generated based on known vulnerability databases [5] and they are widely used in the network security to identify the minimal subset of vulnerabilities to be fixed to prevent all known attacks [6], or to calculate security risk measures (e.g., the probability of a successful attack) [4]. We use AGs for computing the attacker's optimal attack plan to each defender's honeypot allocation action to measure defender's action effectivness.

We present a novel game-theoretic model of security hardening based on attack graphs and a case study analyzing the hardening solutions for sample networks.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

2. NETWORK HARDENING GAME

We model the network hardening problem as a Stackelberg game, where the defender acts first, taking actions to harden the network by adding up to k honeypots (HPs). The attacker is the follower who selects an optimal attack plan based on (limited) knowledge about the defender's strategy. In particular, we assume that the attacker learns the number and type of deployed HPs; however he does not know which specific hosts are HPs and which are real.

A game instance is based on a specific computer network (e.g., in Fig. 1a). A network has a set of host types, 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). The same host types present the same attack opportunities, so they are represented only once in an attack graph. During an attack, a specific host of a given type is selected randomly with uniform probability. E.g., adding more HPs of a specific type increases the likelihood that the attacker who interacts with this host type will choose a HP instead of a real host. If the attacker interacts with a HP during an attack, he is immediately detected and the attack ends. The attacker is rational and maximizes the expected utility taking into account the probabilities of interacting with HPs, his actions' costs and success proba-

bilities, and rewards from successful attacks. Installing and maintaining HPs has a cost for the defender depending on the host type that is duplicated. He minimizes his total expected loss which consists of the expected loss for being attacked and the cost for deploying the HPs into the network. The Stackelberg equilibrium is found by selecting the defender's pure action that minimizes the expected loss under the assumption that the attacker responds with an optimal attack [8]. If the attacker is indifferent between multiple attacks, it is typical to break ties in favor of the defender [8].

Computation of the equilibrium relies on computing the optimal attack policy as described in the following section.

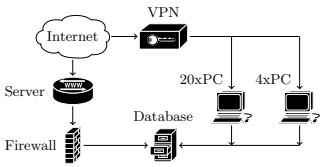
2.1 Attack Graphs and Attack Policies

An attack graph (AG) captures all known ways that the computer network can be compromised. It is an and-or graph consisting of the fact nodes (OR), logical statements about the network (i.e., access to a database), and actions nodes (AND), that change the statements from *false* to *true*.

To characterize the attacker's reaction to the set of honeypots, we compute a full *contingent attack policy* (AP), which defines an action for each situation that may arise during an attack as described in [1]. This allows identifying actions likely to be executed by a rational attacker as well as the *order* of their execution. The attacker chooses the optimal AP that maximizes his expected utility, which is an NP-hard problem [2]. We address this issue by translating AGs into an *Markov Decision Processes* and introducing several pruning techniques that reduce the computation considerably. First, we use a generalized version of the *Sibling-Class Theorem* from [2]. It states that in certain cases the optimal action order can be determined directly from the actions' success probabilities and costs, without any search. Second, we developed a heuristic to compute lower and upper bounds of the expected reward for the AG, which we use in a branch and bound manner to prune out the unpromising subtrees.

3. EXPERIMENTS

As an example of our result, we experimentally evaluated the proposed network hardening game on the network topology in Fig. 1a taken from [3]. This network consists of a server (srv), a vpn, a firewall (fw), a database (db) a group of 20 PCs (20grp) and a group of 4 PCs (4grp). The defender's expected loss (EL) without HPs is 1973. In Fig. 1b we present the optimal HP allocations computed by the game for different numbers of HPs k. For the first two HPs, it is best to duplicate the server and vpn—the network "door"—despite the fact that they are not the most valuable hosts. The lowest EL 617 is reached with 6 HPs by duplicating the database, server and vpn. Any additional HP only increases the EL, because their contribution in detecting the attack does not compensate their maintenance cost, so the cheapest option is selected to minimize the cost.



(a) Network

host	$k=1$	$k=2$	$k=6$	$k=7$
db	0	0	2	2
srv	1	1	2	2
vpn	0	1	2	2
4grp	0	0	0	1
EL	1524	1085	617	635

(b) HP allocations

Figure 1: (a) A small network topology. (b) The optimal HP allocations for 1,2,6 and 7 HPs and corresponding defender's expected loss (EL).

4. CONCLUSION

We introduce a game-theoretic model for the network hardening problem. The defender seeks an optimal deployment of honeypots into the network, while the attacker tries to attack the network and avoid the interaction with the honeypots. Our model provides a novel combination of using compact representation of the strategies of the attacker in the form of attack graphs, and using deception by the defender. By translating the attack graphs into MDPs and employing a number of pruning techniques, we are able to solve problems of realistic size and analyze the results for realistic case studies. We show that a few HPs can significantly reduce the defender's expected loss.

Our work has significant potential for further research. Since the majority of the required input data can be automatically acquired by standard network scanning tools, or extracted from existing vulnerability databases, the proposed model can be deployed in real-world networks and evaluated in practice. Secondly, our model can be further extended from the game-theoretical perspective and use additional uncertainty about the knowledge of the attacker, or model multiple types of the attacker using Bayesian variants of Stackelberg games.

Acknowledgments

This research was supported by the Office of Naval Research Global (grant no. N62909-13-1-N256) and 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.

REFERENCES

[1] K. Durkota and V. Lisý. Computing optimal policies for attack graphs with action failures and costs. In *STAIRS*, pages 101–110, 2014.

[2] R. Greiner, R. Hayward, M. Jankowska, and M. Molloy. Finding optimal satisficing strategies for and-or trees. *Artificial Intelligence*, pages 19–58, 2006.

[3] J. Homer, X. Ou, and D. Schmidt. A sound and practical approach to quantifying security risk in enterprise networks. *Kansas State University*, 2009.

[4] S. Noel, S. Jajodia, L. Wang, and A. Singhal. Measuring security risk of networks using attack graphs. *International Journal of Next-Generation Computing*, 1(1):135–147, 2010.

[5] X. Ou, W. F. Boyer, and M. A. McQueen. A scalable approach to attack graph generation. In *CCS*, pages 336–345, 2006.

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

[7] L. Spitzner. *Honeypots: tracking hackers*. Addison-Wesley Reading, 2003.

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

Optimal Network Security Hardening Using Attack Graph Games

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

Abstract

Preventing attacks in a computer network is the core problem in network security. We introduce a new game-theoretic model of the interaction between a network administrator who uses limited resource to harden a network and an attacker who follows a multi-stage plan to attack the network. The possible plans of the attacker are compactly represented using attack graphs, while the defender adds fake targets (honeypots) to the network to deceive the attacker. The compact representation of the attacker's strategies presents a computational challenge and finding the best response of the attacker is NP-hard. We present a solution method that first translates an attack graph into an MDP and solves it using policy search with a set of pruning techniques. We present an empirical evaluation of the model and solution algorithms, evaluating scalability, the types of solutions that are generated for realistic cases, and sensitivity analysis.

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 attacks.

We focus on network security hardening problems in which a network administrator (defender) reduces the risk of attacks on the network by introducing honeypots (fake hosts or services) into their network [Qassrawi and Hongli, 2010]. Legitimate users do not interact with honeypots; hence, honeypots act as decoys and distract attackers from the real hosts, send intrusion detection alarms to the administrator, and/or gather detailed information the attacker's activity [Provos, 2004; Grimes *et al.*, 2005]. However, believable honeypots are costly to set up and maintain. For example, in [Carroll and Grosu, 2011; Cai *et al.*, 2009] the authors propose a game-theoretic model that studies various camouflaging signals that honeypots can send to the attacker in order to minimize the chance of being detected. Deciding how to optimally allocate honeypots to reduce the risk of attacks on a network presents a challenging decision for the defender. On the other hand, a well-informed attacker should anticipate the use of honeypots and try to avoid them.

We use game theory to model this adversarial interaction and to determine the best way to use honeypots against a well-informed attacker. We introduce a novel game-theoretic model of network hardening using honeypots that extends the existing line of Stackelberg security games [Tambe, 2011] by combining two elements: (1) we adopt a compact representation of strategies for attacking computer networks called *attack graphs*, (2) the defender uses *deception* instead of directly allocating resources to harden targets.

Attack graphs (AGs) can 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. However, finding the optimal attack plan in an attack graph is an NP-hard problem [Greiner *et al.*, 2006]. We address this issue by translating attack graphs into an MDP and introducing a collection of pruning techniques that reduce the computation considerably.

Deploying honeypots changes the structure of the network and increases uncertainty for the attacker. In this game model we assume that the attacker knows the number of deployed honeypots and their type (e.g., a database server). However, the attacker does not know which specific hosts are honeypots and which are real. While the assumption that the attacker knows the number/type of honeypots is strong, it corresponds to a worst-case, well-informed attacker. Our model could also be extended to include uncertainty about these variables, but it would further increase the computational cost of solving the model.

We present five main contributions: (1) a novel game-theoretic model of security hardening based on attack graphs, (2) algorithms for analyzing these games, including fast methods based on MDPs for solving the attacker's planning problem, (3) a case study analyzing the hardening solutions

for sample networks, (4) empirical evaluation of the computational scalability and limitations of the algorithms, and (5) sensitivity analysis for the parameters used to specify the model.

2 Network Hardening Game Using Honeypots

In this section we introduce a game-theoretic model for the network hardening problem. Our model is a Stackelberg game, where the defender acts first, taking actions to harden the network by adding honeypots (HPs). The attacker is the follower that selects an optimal attack plan based on (limited) knowledge about the defender's strategy. In particular, we assume that the attacker learns the number and type of HPs added to the network, but *not* which specific hosts are real and fake.

An instance of the game is based on a specific computer network like the one shown in Fig. 1c (based on [Homer *et al.*, 2009]). A network has 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. During an attack, a specific host of a given type is selected randomly with uniform probability.

More formally, a computer network $y \in \mathbb{N}^T$ contains y_t hosts of type $t \in T$. The defender can place up to k honeypots into the network y, so his actions are represented by $x \in X \subset \mathbb{N}_0^T$ with $\sum_{t \in T} x_t \leq k$, specifying that x_t hosts type $t \in T$ will be added to the network as honeypots (e.g., by duplicating the configurations of the real hosts with obfuscated data). The modified network consists of $z_t = x_t + y_t$ hosts of type t. Adding more HPs of a specific type increases the likelihood that the attacker who interacts with this type of host will choose a HP instead of a real host. If the attacker interacts with a HP during an attack, he is immediately detected and the attack ends. The attacker is rational and maximizes the expected utility taking into account the probabilities of interacting with HPs, his actions' costs and success probabilities, and rewards from successful attacks. He selects his attack strategy from set Ξ defined later. Installing and maintaining HPs has a cost for the defender depending on the host type ($c(t)\ t \in T$) that is duplicated. The defender minimizes his total expected loss l which consists of (1) the expected loss for the attacked hosts and (2) the cost for adding the HPs into the network. The Stackelberg equilibrium is found by selecting the pure action of the defender that minimizes the expected loss under the assumption that the attacker will respond with an optimal attack [Conitzer and Sandholm, 2006]. If the attacker is indifferent between multiple attacks, it is typical to break ties in favor of the defender [Tambe, 2011]. The defender's action is

$$x^* = \operatorname*{argmin}_{x \in X} \{ l(x, \operatorname*{argmax}_{\xi \in \Xi} \{ \mathbb{E}(\xi, x) \}) \}. \quad (1)$$

The minimization over all defender actions is performed by systematically checking each option; however, the main computation burden of computing the optimal attack strategy is substantially reduced by caching and reusing results of subproblems that are often the same. Computation of this equilibrium relies on computing the optimal attack policy as explained in the following section.

3 Attack Graphs and Attacker's Strategies

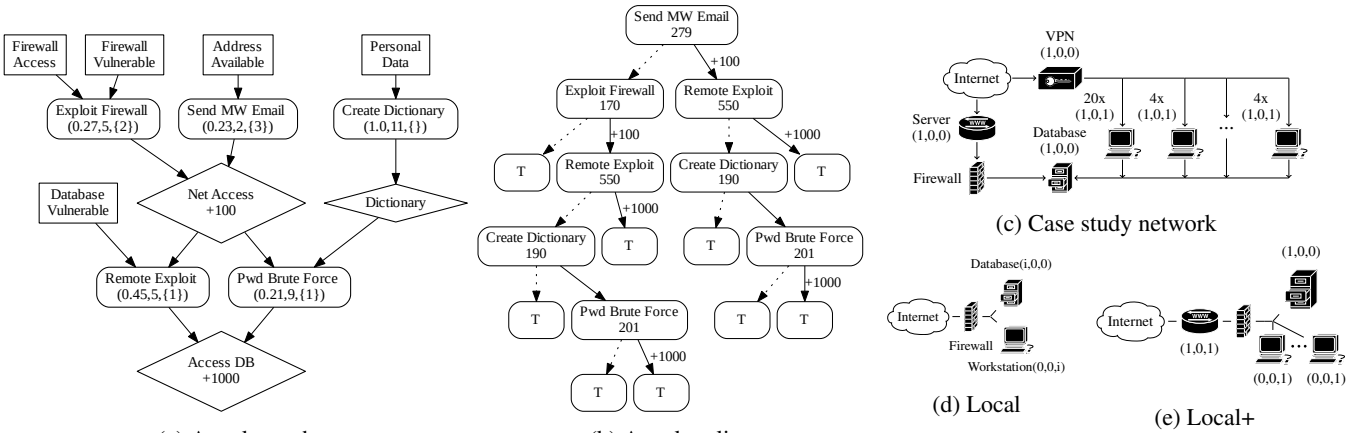
There are multiple representations of attack graphs common in the literature. We use *dependency attack graphs*, which are more compact and allow more efficient analysis than the alternatives [Obes *et al.*, 2013]. Fig. 1a is an example attack graph with high-level actions for illustration. Formally, it is a directed AND/OR graph consisting of fact nodes F (OR) and action nodes A (AND). Every action has *preconditions* ($\text{pre}(a) \subseteq F$) – a set of facts that must be true before the action can be performed – and *effects* ($\text{eff}(a) \subseteq F$) – a set of facts that become true if the action is successfully performed. These relations are represented by edges in the attack graph. 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.

Every action has associated a probability of being performed successfully $p_a \in (0,1]$, and cost $c_a \in \mathbb{R}^+$ that the attacker pays regardless of whether the action is successful. The costs represent the time and resources for the attacker to perform the action. Finally, every action a interacts with a set of host types $\tau_a \subseteq T$. 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. Each fact $f \in F$ has associated an immediate reward $r_f \geq 0$ that the attacker receives when the fact becomes true (e.g., an attacker gains reward by gaining access to a particular host or compromising a specific service). At any time we allow the attacker to terminate his attack by a stop action denoted T.

An illustrative example of attack graph is depicted in Fig. 1a. Diamonds and rectangles are fact nodes that are initially *false* and *true*. Actions (rounded rectangles) are denoted with a label and a triple (p_a, c_a, τ_a). The attack proceeds in a top-down manner. At the beginning the attacker can perform actions *Exploit-Firewall*, *Send-MW-Email* or *Create-Dictionary*. The action *Exploit-Firewall*'s preconditions are {*Firewall-Access*, *Firewall-Vulnerable*} and its effect is {*Net-Access*}. If this action is performed, the attacker immediately pays cost $c_a = 5$, interacts with host types $\tau_a = \{2\}$, and with probability $p_a = 0.27$ the action's effects become true. In that case the attacker obtains reward +100.

Attack graphs can be automatically generated by various tools. We use the MulVAL [Ou *et al.*, 2005], which constructs an 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 the information about the costs and success probabilities for different actions can be estimated using the Common Vulnerability

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



(a) Attack graph (b) Attack policy (c) Case study network (d) Local (e) Local+

Figure 1: (a) Attack graph representing the possible ways of gaining an access to the database. (b) Optimal attack policy for the attack graph in (a); (c,d,e) network topologies with triples (r,l,c) at hosts denoting number of remote (r), local (l) and client (c) vulnerabilities at that host.

Scoring System [Mell *et al.*, 2006] values available in the National Vulnerability Database [Bacic *et al.*, 2006], historical data, red team exercises, or be directly specified by the network administrator.

Optimal attack policy In order to fully characterize the attacker's reaction to the set of honeypots, we need to 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. This is necessary to evaluate effectiveness of placing honeypots or any other intrusion detection sensors. 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.

The attack strategies Ξ are all contingent plans consistent with the attack graph. The optimal attack policy maximizes attacker's expected utility and in case of ties favors the defender. Fig. 1b depicts the optimal attack policy for the attack graph in Fig. 1a without any honeypots. Nodes represent suggested actions to perform with their expected rewards if strategy is followed. The first action suggested by this policy is to *Send-MW-Email*. If the action is successful, the attacker immediately obtains reward +100 for reaching *Net-Access* and follows the right (sub)policy (solid arc). If *Send-MW-Email* fails, attacker's best choice is to perform *Exploit-Firewall* and expect reward 170. The attack terminates (T) if there are no actions to perform or the expected rewards do not surpass the costs of the actions.

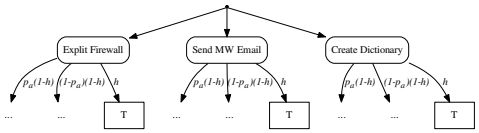
4 Solving AG using MDP

In the Stackelberg game model we assume that the attacker observes the defender's strategy, which can be used in computing the best-response attack strategy. We represent the problem of computing the attacker's best response as a finite horizon Markov Decision Process (MDP) [Bellman, 1956]. The MDP for attack graph AG is defined as a tuple $\langle B, S, P, \rho \rangle$, where: (1) $B = A \cup \{\mathrm{T}\}$ is the set of actions, (2) S is set of states $s = (\alpha_s, \phi_s, \tau_s)$, where: $\alpha_s \subseteq B$ is the set of actions that a rational attacker can still perform, $\phi_s \subseteq F$ is the set of achieved facts, and $\tau_s \subseteq T$ is the set of host types that the attacker has interacted with so far. The initial MDP state is $s_0 = (B, \emptyset, \emptyset)$. (3) $P^a_{ss'} \in (0,1]$ is the probability that performing action a in state s leads to state s'. Performing action a can lead to one of three possible outcomes: either (i) a interacts with a honeypot with probability $h = 1 - \prod_{t:\tau_a \setminus \tau} (\frac{z_t - x_t}{z_t})$ and his attack immediately ends; (ii) action a does not interact with a honeypot and is successful with probability $p_a(1-h)$ or (iii) action a does not interact with a honeypot and neither is successful with probability $(1-p_a)(1-h)$.The sets defining the newly reached state are updated based on these outcomes. Finally, (4) $\rho^a_{ss'}$ is the attacker's reward for performing action a in state s leading to s'. It is based on the set of facts that became true, the action cost c_a and interaction with a honeypot.

We compute the optimal policy in this MDP using backward induction based on depth-first search, with several enhancements to speed up the search. A major performance enhancement is *dynamic programming*. Since the same states in the MDP can often be reached via more than one sequence of actions, we cache the expected rewards and corresponding policies of the visited states and reuse it when possible. In addition we use the following pruning techniques.

Sibling-Class Pruning

In this section we introduce the Sibling-Class Theorem (SCT) and its use to prune the search tree. This theorem states that in some cases, the optimal order for executing actions can be determined directly without search. It was proved in [Greiner *et al.*, 2006] in the context of "probabilistic AND-OR tree resolution" (PAOTR). In [Buldas and Stepanenko, 2012], the AND part of the theorem is proven in the context of simplified attack graphs. Both works assume that actions have no preconditions, i.e., the inner nodes of the AND/OR tree represent only the conjunctions and disjunctions and do not have



(a) Decision point

Figure 2: An MDP decision point for AG from Fig. 1a.

any costs or probabilities of success. Moreover, the theorem was proven only for the special case of trees not for directed acyclic graphs. We generalize the SCT to handle the more general case that we need for our attack graphs.

Actions $a, b \in A$ belong to the same *OR-sibling class* iff they have the exact same effects. Actions $\alpha \subseteq A$ belong to the same *AND-sibling class* iff there is a "grandchild" action $g \in A$ that can be executed iff all of the actions in α are successful and none of the actions has an effect that would help enable other actions, besides g. We define the *R-ratio* of actions in sibling classes as R(a) = $\frac{p_a}{c_a}$ if a belongs to an OR-sibling class; and R(a) = $\frac{1-p_a}{c_a}$ if a belongs to an AND-sibling class.

Theorem 1 (Sibling class theorem) *Let ξ be the optimal attack policy for the attack graph AG. Then for any actions x, y from the same sibling class, such that $R(y) > R(x)$, x is never performed before y in ξ.*

Intuitively, in OR-sibling class, it is reasonable to start with the high probability low cost actions. In AND-sibling class, it is reasonable to start with cheap actions that have low success probability to fail fast and avoid paying unnecessary costs. Due to limit constraints, the theorem is formally proven in the extended version of this paper. Unfortunately, actions that can interact with honeypots violate the monotonicity property assumed in the proof of SCT. These actions cannot be pruned neither preferred to other actions. Thus, we prune only actions that do not interact with honeypots, belong to exactly one sibling class and there is another action in the same sibling class with higher R-ratio.

Branch and Bound

We compute lower (LB) and upper (UB) bounds of the expected reward in each MDP state. Consequently, we use them to prune the MDP subtrees if they are provably suboptimal.

Lower bounds are computed from the values of the previously computed MDP subtrees and bound the future MDP subtrees. First, we present necessary properties of the optimal attack policy.

Proposition 1 *Let ξ be the optimal attack policy in state s starting with action a, ξ_+ and ξ_- be its (sub)policies if action a succeeds or fails, respectively, and $\mathbb{E}[\xi]$ be the value of the policy. Then, (i) $\mathbb{E}[\xi_+] + \rho - c_a/p_a \geq \mathbb{E}[\xi_-]$, where $\rho = \sum_{f:\text{eff}(a)\setminus\phi} r_f$ is an immediate reward and (ii) $\mathbb{E}[\xi] \geq \mathbb{E}[\xi_-]$.*

In Fig. 2a is the first decision point for the attack graph in Fig. 1a, where the attacker decides among three available actions: (*Exploit-Firewall*, *Send-MW-Email* or *Create-Dictionary*). In every decision point of the MDP's depth-first search tree we explore each of its subtrees one by one and use maximal value M of the previous actions as a lower bound value for next action. Since every action's outcome (success or fails) is explored separately, we bound them individually. Action a's successful branch is bounded by $M - \rho + c_a/p_a$, which results from Prop. 1(i). Its false branch is bounded by $\frac{M+c_a-p_a(\mathbb{E}[a^+]+\rho)}{1-p_a}$, where $\mathbb{E}[a^+]$ is the action a's successful branch's expected reward, which results from Prop. 1(ii). Extended version of the paper contains the proof and bound derivations.

We compute the **upper bound** as the expected reward of the optimal policy in an attack graph relaxed in two ways. First, we ignore the action costs ($c_a = 0$) and the probabilities of touching HPs ($h = 0$). This only improves the expected utility of the attack and causes the optimal strategy to always use all available actions. Therefore, we can compute the expected reward of the policy by computing the probability that each fact is achieved if the attacker attempts all actions. In order to compute this probability efficiently, we run a depth first search in the attack graph from each fact which provides a reward, traversing edges only in the opposite direction. Moreover, to avoid complications caused by cycles, we disregard edges leading to AND nodes from already traversed parts of the attack graph. Removing edges to AND nodes can always only increase the probability of reaching the reward fact.

5 Experiments

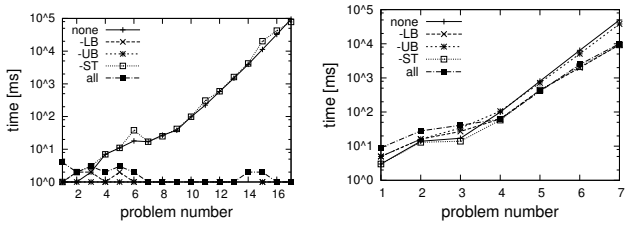
The experiments analyze the algorithm for optimal attack policy computation and evaluate game models for network hardening problems. All experiments were run on one core of Intel i7 3.5GHz processor with 10GB memory limit. Attack graphs for the experiments were generated with MulVAL [Ou *et al.*, 2005] tool, augmented with additional cost and probability information.

Network topologies for experiments are depicted in Fig. 1c (Main-*i*), 1d (Local-*i*) and 1e (Local+*i*), where *i* is: (i) number of workstation client vulnerabilities in Local-*i* , (ii) number of workstations in Local+*i* and (iii) number of workstation groups in Main-*i*. Host types are labeled with (r,l,c), where *r*,*l*,*c* denotes the numbers of *remote*, *local* and *client* vulnerabilities for that type. The topology of the network and the number and type of vulnerabilities determines the number of actions in the attack graph.

5.1 Optimal Attack Planning

Pruning Techniques

In this section we evaluate the contribution of the pruning techniques: Sibling Theorem (ST), lower bound (LB), and upper bound (UB) for the branch and bound. Since techniques may prune the same branches, we measure each technique's contribution by measuring the algorithms slowdown after disabling it. Thus, we compare 5 settings: *none* (no techniques is used), *all* (all techniques are used), *-ST* (all but ST), *-LB* (all but LB) and *-UB* (all but UB). Fig. 3 shows optimal attack policy computation times of networks Local-*i* and Loca+*i* using each technique setting. In Local-*i* network we add vulnerabilities to the workstation, creating large decision points. However, ST can effectively prune them and choose the best action (compare *all* to *-ST*). In Local+*i* network the



(a) Local-i (b) Local+i

Figure 3: Time comparison of the pruning techniques. Legend: none - no pruning technique, (-LB) - all but lower bound, (-UB) - all but upper bound, (-ST) - all but Sibling Theorem, (all) - all pruning techniques.

Problem	# A	DP	UID	SPUDD
Local3	9	<1	<1	<1
Local11	25	<1	(OoM)	3
Local19	41	<1	(OoM)	3348
Local+3	16	<1	71	1
Local+6	28	**3**	(OoM)	125
Local+8	36	**406**	(OoM)	1951

(a)

host	1	2	3	5
db	0	0	1	2
srv	1	1	1	2
vpn	0	1	1	1
1grp	0	0	0	0
2grp	0	0	0	1

(b)

Table 1: (a) Computation times (in seconds) of computing optimal attack policies by DP, GUIDO and SPUDD. (OoM) - Out of Memory, (Err) - planner exited with segmentation error. (b) Optimal allocation of different number of HPs (columns) to the host types (rows) in the Main-7 network.

branch and bound helps the most (*-LB* and *-UB*). In the remaining of experiments, we include all techniques since they do not have negative impact and can dramatically speed up the computation. We refer to the proposed attack planning algorithm as DP (dynamic-programing).

Comparison with Other Algorithms

We compare DP to other two approaches that can compute the optimal policy for an attack graph. The first method converts the AG it to an *Unconstrained Influence Diagram* (UID) as described in [Lisý and Píbil, 2013] and uses GUIDO [Isa *et al.*, 2007] to solve it. The second approach translates AG to a probabilistic planning domain problem, which is then solved by SPUDD [Hoey *et al.*, 1999]. It uses iterative value computation and was 3rd best planner in 2011 at the International Planning Competition (IPC) in the MDP track. We chose SPUDD because it guarantees to return an optimal contingency policy, while the first two planners from IPC do not. In Tab. 1(a) we present computation times of the algorithms. Our algorithm dramatically outperforms the other two approaches, where the hardest network Local+8 was solved 5x faster than using SPUDD planner.

5.2 Network Hardening Game Analysis

In this section we evaluate the network hardening. We focus on a typical small business network topology depicted in Fig. 1c (Main-i) and we find honeypot deployment that minimizes the defender's loss in this network. Attacker's actions costs in AG are set randomly between 1 and 100. The rewards for compromising the host types are 1000 for workstations (ws), 2000 for vpn, 2500 for server (srv) and 5000 for database (db). For simplicity, we assume that the defender values (l_t) them the same as the attacker. However, this payoff restriction is not required in our model. The defender pays $c_{hp}(t) = \gamma l_t$ for adding honeypot of type t, where $\gamma \in [0,1]$ is parameter we alter. It corresponds to the fact that more valuable hosts are generally more costly to imitate by honeypots.

Scalability Experiment

We analyze the scalability of our game model to determine the size of the networks that can reasonably be solved using our algorithms. For each network Main-i, we create a game model and find Stackelberg equilibrium. In Fig. 4a we present computation times (note log-scale y-axis) to solve networks Main-6 through Main-8 (individual plots), given the number of HPs that the defender can deploy. The algorithm scales exponentially with both parameters. However, we note that typical real-world cases will have few honeypots.

5.3 Case-Study

In this section we examine in detail the network Main-7 with $\gamma = 0.02$. We analyze the defender's loss dependence on the number of honeypots and his relative regret for modeling the attack graph inaccurately (e.g., if he estimates the action costs, action probabilities or rewards incorrectly).

Defender's loss

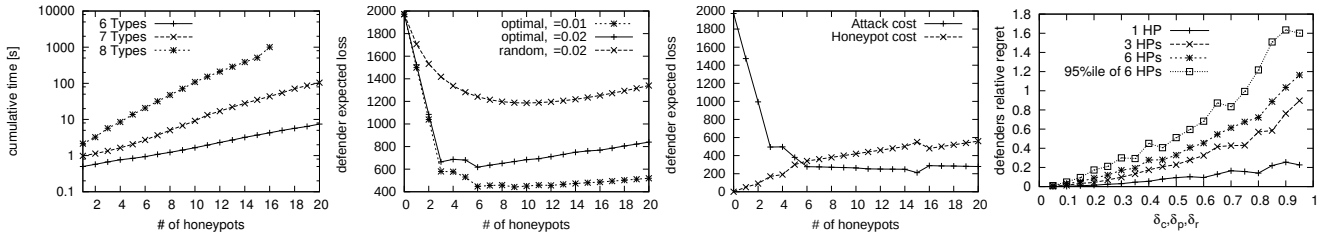
Using a large number of HPs is not necessarily beneficial for the defender as they may cost more than they contribute to protecting the network. The defender's expected loss using different number of the HPs is shown in Fig. 4b (see "optimal, $\gamma = 0.02$"), where it is optimal to deploy 6 HPs for $\gamma = 0.02$ and for 9 HPs for the cheaper $\gamma = 0.01$ setting. In the same figure, we also present the defender's loss for deploying random honeypots instead the optimal suggested by our model, which is roughly twice as bad. We also note how dramatically merely 3 HPs can decrease the expected loss using the game-theoretic approach.

In Fig. 4c are separate parts of the defender's loss: the expected loss for having the hosts attacked and the costs for deploying the HPs. Interestingly, with 16 HPs the defender decides to use less expensive HPs and put the network in higher risk, since this choice results in a lower overall expected loss.

In Tab. 1(b) we present the defender's optimal HP types allocations (rows) for a given total number of deployable HPs (columns). I.e., with one HP, it is best to duplicate the server (srv). The server is not the most valuable type (2500 while the database is worth 5000). However, it is a bottleneck of the network; protecting the server partially defends both the server and the database, rather than only database. With two HPs it is best to duplicate the server and VPN (again, the bottlenecks). Only with the 3rd HP does the defender duplicate the database.

Sensitivity Analysis

Computing the defender's optimal strategy heavily relies on the Attack Graph structure and knowledge of the action costs, probabilities and rewards, which the defender can only approximate in a real-world scenarios. In the following experiments we analyze the sensitivity of the defender's strategy and utility to inaccurate approximation of the attack graphs.



(a) Scalability (b) Defender's loss (c) Defender's costs (d) Sensitivity

Figure 4: (a) Time comparison of solving the game for various networks and the number of HPs; (b) Defender's expected loss given the γ and allocating HPs optimally and randomly. (c) Individual parts of the defender's loss: the expected cost of having the network attacked (Attack cost) and cost of deploying the HPs (Honeypot cost). (d) Defender relative regret given the perturbation of action costs, success probabilities and rewards in networks with 1, 3 and 6 HP, and 95th percentile for 6 HPs.

We use the following notation: defender's strategy $\bar{d}$ (resp. loss $\bar{l}_{\bar{d}}$) is computed from the attack graph known to the defender, which imprecisely approximates the attack graph truly describing attacker's behavior; d (resp. loss l_d) is the optimal strategy based on the true attack graph according which the attacker really behaves; and $l_{\bar{d}}$ is the defender's loss when strategy $\bar{d}$ is used against the real attack graph.

In the sensitivity analysis we compute the *relative regret* as $regret = |l_{\bar{d}} - l_d|/\bar{l}_{\bar{d}}$, which is the defender's relative loss ratio for not knowing the attacker's real attack graph. In other words, if the defender knew the true attack graph model and acted optimally, this is how much he would gain. Thus, $regret = 0$ means he has no regret and $regret = 1$ means that his loss is by 100% greater than it could have been.

In this experiments the attack graph action probabilities and costs were chosen randomly from the intervals $p_a \in [0,1]$, $c_a \in [0,200]$ in order not to depend on the specific initial attack graph values provided by MulVAL. To generate the imprecise attack graph available to the defender, we perturbed the generated attack graph based on values $(\delta_p, \delta_c, \delta_r) \in [0,1]^3$, where each value represents the limit on the size of the perturbation on action probabilities (δ_p), costs (δ_c) and fact rewards (δ_r). The perturbed attack graph is obtained as follows: for each action $a \in A$ the perturbed probability $\bar{p}_a$ is chosen from an interval $[p_a - \delta_p, p_a + \delta_p]$ restricted to $[0.05, 0.95]$ to prevent them becoming impossible ($\bar{p}_a = 0$) or infallible ($\bar{p}_a = 1$); perturbed cost $\bar{c}_a$ is chosen from $[c_a(1-\delta_c), c_a(1+\delta_c)]$; and for each fact $f \in F$, the perturbed fact reward is $\bar{r}_f \in [r_f(1-\delta_r), r_f(1+\delta_r)]$, where the values are selected uniformly within the given intervals. 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 modeling them, which cannot be assumed for probabilities.

In our experiments we perturbed the each component from 0.05 to 0.95 by 0.05 steps and measured the defender's loss. The results presented are mean values computed from 100 runs. In Fig. 4d we present the defender's relative regret for increasing error perturbations for deploying 1, 3 and 6 HPs, where we perturbed all three perturbation components. The results show the solutions are fairly robust, i.e., assuming that the defender models all components inaccurately by at most 30% ($\delta_c = \delta_p = \delta_r = 0.3$) in scenario with 6 HPs, his HP deployment choice could have been better off by only 17%. We also examined the effect of perturbing each component individually, however, due to the space limit, we did not include figures (which are similar). Out of the three components, the defender's strategy was most sensitive to action probability perturbations, where the relative regret reaches value 0.8 for the case when $\delta_p = 0.95$ for 6 HPs. The sensitivity to reward perturbations reached relative regret loss of about 0.3 for $\delta_r = 0.95$. Results were least sensitive to action cost perturbations, resulting in relative regret loss below the value 0.002 even for $\delta_c = 0.95$. Fig. 4d also presents the 95th percentile of the regret values for the case with 6 HPs to show the robustness in extreme cases.

Note that with an increasing number of HPs, the defender's relative regret grows (e.g., Fig. 4d), however, this can be misleading. Further data analysis revealed that the actual *absolute regrets* $|l_{\bar{d}} - l_d|$ for 3 HPs (318) and 6 HPs (340) are very similar. The reason why relative regret seem to grow with the increasing number of HPs is due to the decrease of the observed loss $\bar{l}_{\bar{d}}$ (denominator it relative regret formula). The observed loss $\bar{l}_{\bar{d}}$ for 3 HPs is 1994, while for 6 HPs 1049, which shows that the relative regret does not seem to depend on the number of HPs.

6 Conclusion

We introduce a game-theoretic model for the network hardening problem. The defender seeks an optimal deployment of honeypots into the network, while the attacker tries to attack the network and avoid the interaction with the honeypots. Our model provides a novel combination of using compact representation of the strategies of the attacker in the form of attack graphs, and using deception by the defender. By translating the attack graphs into MDPs and employing a number of pruning techniques, we are able to solve problems of realistic size and analyze the results for realistic case studies. Moreover, we showed that our model produces robust solutions even if the input data are imprecise.

Our work has significant potential for further research. Since the majority of the required input data can be automatically acquired by standard network scanning tools, or extracted from existing vulnerability databases, the proposed

model can be deployed in real-world networks and evaluated in practice. Secondly, our model can be further extended from the game-theoretical perspective and use additional uncertainty about the knowledge of the attacker, or model multiple types of the attacker using Bayesian variants of Stackelberg 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. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated. Viliam Lisý is a member of the Czech Chapter of The Honeynet Project.

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.

[Bellman, 1956] Richard Bellman. Dynamic programming and lagrange multipliers. *PNAS*, 42(10):767, 1956.

[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.

[Buldas and Stepanenko, 2012] Ahto Buldas and Roman Stepanenko. Upper bounds for adversaries' utility in attack trees. In *GameSec*, pages 98–117. 2012.

[Cai *et al.*, 2009] Jin-Yi Cai, Vinod Yegneswaran, Chris Alfeld, and Paul Barford. An attacker-defender game for honeynets. In *Computing and Combinatorics*, pages 7–16. Springer, 2009.

[Carroll and Grosu, 2011] Thomas E Carroll and Daniel Grosu. A game theoretic investigation of deception in network security. *Security and Communication Networks*, 4(10):1162–1172, 2011.

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

[Greiner *et al.*, 2006] Russell Greiner, Ryan Hayward, Magdalena Jankowska, and Michael Molloy. Finding optimal satisficing strategies for and-or trees. *Artificial Intelligence*, pages 19–58, 2006.

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

[Hoey *et al.*, 1999] Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. Spudd: Stochastic planning using decision diagrams. In *UAI*, pages 279–288, 1999.

[Homer *et al.*, 2009] John Homer, Xinming Ou, and David Schmidt. A sound and practical approach to quantifying security risk in enterprise networks. *Kansas State University Technical Report*, pages 1–15, 2009.

[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.

[Isa *et al.*, 2007] Jiri Isa, Viliam Lisy, Zuzana Reitermanova, and Ondrej Sykora. Unconstrained influence diagram solver: Guido. In *IEEE ICTAI*, pages 24–27, 2007.

[Lisý and Píbil, 2013] Viliam Lisý and Radek Píbil. Computing optimal attack strategies using unconstrained influence diagrams. In *PAISI*, pages 38–46. 2013.

[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 *Proceedings of the 2004 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.

[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 *Networks Security Wireless Communications and Trusted Computing (NSWCTC), 2010 Second International Conference on*, volume 2, pages 462–467. IEEE, 2010.

[Sawilla and Ou, 2008] Reginald E. Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. In Sushil Jajodia and Javier Lopez, editors, *Computer Security - ESORICS 2008*, volume 5283 of *Lecture Notes in Computer Science*, 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.

A Proofs

This appendix contains proofs of the propositions from the "Optimal Network Security Hardening Using Attack Graph Games" submitted to IJCAI 2015.

A.1 Sibling-Class Theorem for Attack Graphs

In this section we proof that Sibling Class Theorem may be used for Attack Graphs.

Theorem 2 (Sibling class theorem) *Let ξ be an optimal attack policy for the attack graph AG. Then for any actions x, y in the same sibling class, such that $R(y) > R(x)$, x is never performed before y in ξ.*

The proofs for the AND and OR sibling classes are symmetric; hence, we focus only on the OR sibling class. The proof of this theorem in [Greiner *et al.*, 2006] relies on a lemma about attack policies. If we denote a tree rooted in action x with subtree ξ_+ (when x is successful) and subtree ξ_- (when action fails) by $(x;\xi_+;\xi_-)$ then the lemma can be rephrased in the following way:

Lemma 1 *Let $x, y \in A$ be actions in the same OR-sibling class of an attack graph such that $R(y) > R(x)$. Let ξ_{xy} be an optimal policy for the attack graph in the form $(x;\xi_+;(y;\xi_+;\xi_-))$, then $\xi_{yx} = (y;\xi_+;(x;\xi_+;\xi_-))$ is a valid policy with a higher expected reward than ξ_{xy}.*

The proof of the Sibling class theorem in [Greiner *et al.*, 2006] works by induction on the number of attack actions in the optimal policy. It is demonstrated on Figure 5 taken form the article. For contradiction, assume that there are actions x and y in the wrong order in the optimal attack policy. Without loss of generality, assume a case with an action x in the root of the policy and its sibling y at several places in the negative branch of x (Figure 5(a)). The positive branch does not include y, because it is an OR-sibling of x, which means that the fact achieved by y is already achieved by x, if it is successful. The proof shows that the expected reward of the policy is not decreased if it starts as the negative branch of x and action x is executed just before action y would be executed in the original policy (Figure 5(b)). Then the proof uses the Lemma 1 to show that if each instance of the actions x and y are switched (Figure 5(c)), the expected reward of the policy is increased if the R-ratio of y is higher then R-ration of x. We argue that a very similar proof is applicable for the more general case of attack graphs defined above. The differences between the model in [Greiner *et al.*, 2006] and our model are:

i) The attack graph is in general a cyclic graph and not a tree.

ii) The attack graph has actions with a probability of success and costs in the inner nodes, and not only in the leaves. This creates preconditions for the actions.

iii) Different attacks give different rewards and it is not necessary to resolve the root node of the attack graph.

iv) The attack terminates when the expected cost of continuing the attack is higher than the expected reward, not when the root of the AND-OR tree is resolved.

The proof is based on properties of the optimal policy with very little reference to the structure of the AND-OR graph. The only use of the AND-OR graph is to assure that after the transformations, the policy is still a legal policy. From the definition of the sibling classes, any action that belongs to a sibling class is a leaf of the attack graph. It does not have any preconditions and it can be executed at any place in the policy. The inner nodes of the attack graph are not moved in the tree and the transformations preserve any preconditions the following actions might have. This means that properties (i) and (ii) do not have any effect on validity of the proof.

The policy in [Greiner *et al.*, 2006] does not have any rewards, only costs for the actions. The proof assumes that each subtree of the policy has an expected cost of execution, but it never requires the costs of the sub-trees to be positive. We can define the expected reward (or negative cost) of a leaf node T as the sum of the rewards acquired in the attack branch from the root to the leaf. Afterwards, we can treat this leaf as any other subtree in the proof. This resolves difference (iii).

The last difference between the models is caused by changing the point when the attack stops. The attack stops at node T if the expected value of the optimal sub-policy continuing from this node is negative. This is a solely a property of the successors of a node in the policy tree. The transformations used in the proof do not change which actions can be executed below a current leaf in the policy and they do not change what other facts can be activated below the current leaves. Therefore, it cannot be optimal to prolong any branch after the transformations. Furthermore, no attack would be terminated earlier due to the transformations. The attacker stops an attack if the expected reward of some subtree would become negative. The proof initially assumes an optimal strategy and shows that the expected reward of the strategy stays the same when x is moved just before y. If the expected cost of playing the subtree rooted at the new position of x were positive, stopping the attack at that place would increase the reward of the whole strategy, which contradicts the optimality of the original strategy. The same holds after switching the positions of x and y. The original proof states that the reward is increased when the two are switched. If in addition, some of the subtrees would have negative expected reward after the switch. Removing the subtree would increase the expected reward even more, which still means that the original strategy was suboptimal.

A.2 Propositions and Lower Bounds

In this section we proof used propositions and lower bounds. We begin with the necessary propositions.

Proposition 2 *Let ξ be an optimal attack policy starting with action a, $\mathbb{E}[\xi]$ be the expected cost of the policy ξ and $\mathbb{E}[\xi_+]$ (resp. $\mathbb{E}[\xi_-]$) be the expected cost of subpolicy ξ after action a succeeds (resp. fails). Then $\mathbb{E}[\xi] \geq \mathbb{E}[\xi_-]$.*

Proof 1 *The first part we prove by contradiction. Assume that $\mathbb{E}[\xi] < \mathbb{E}[\xi_-]$. Then, due to the monotonicity property the attacker could have followed the (sub)policy ξ_- before performing action a, which would have saved him the cost of the action c_a. This new policy would have had higher expected value than the policy ξ, which contradict the optimality of ξ.*

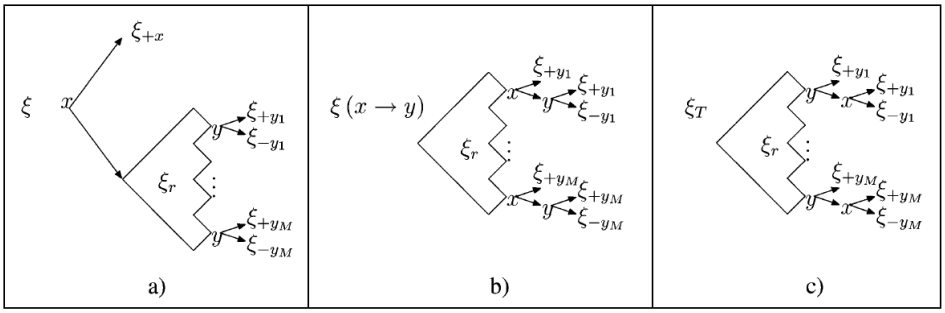


Figure 5: Illustration of the proof of the sibling theorem

Proposition 3 *Let ξ be the optimal attack policy starting with action a, $\mathbb{E}[\xi]$ be the expected cost of the policy ξ and $\mathbb{E}[\xi_+]$ (resp. $\mathbb{E}[\xi_-]$) be the expected cost of subpolicy ξ after action a succeeds (resp. fails). Then $\mathbb{E}[\xi_+] + \rho - c_a/p_a \geq \mathbb{E}[\xi^-]$, where $\rho = \sum_{f:f_a \setminus \phi} r_f$ is an pure immediate reward received if action a succeeds.*

Proof 2 *For convenience, we denote $\bar{p_a} = 1 - p_a$.*

$$\begin{aligned}
\mathbb{E}[\xi] &= -c_a + p_a(\mathbb{E}[\xi_+] + \rho) + \bar{p_a}\mathbb{E}[\xi_-] \\
\textit{Prop. 2} \Rightarrow \mathbb{E}[\xi_-] &\leq -c_a + p_a(\mathbb{E}[\xi_+] + \rho) + \bar{p_a}\mathbb{E}[\xi_-] \\
p_a\mathbb{E}[\xi_-] &\leq -c_a + p_a(\mathbb{E}[\xi_+] + \rho) \\
\mathbb{E}[\xi_-] &\leq -c_a/p_a + \mathbb{E}[\xi_+] + \rho.
\end{aligned}$$

Lower Bounds

In a decision point, the MDP search explores subtrees of all applicable actions one by one. As each action subtree's expected reward is computed, we can their results to bound the future actions' subtrees.

Proposition 4 *Assume the actions $a_1, \ldots, a_{k+1} \in A$ to be in the same decision point in state $s = (\alpha, \phi, \tau)$ of the depth-first search tree for MDP from which the subtrees of actions $a_1, \ldots, a_k$ have been explored with their maximal expected reward M. Let ξ be the optimal attack policy starting with action a_{k+1}, $\mathbb{E}[\xi]$ be the expected cost of the policy ξ and $\mathbb{E}[\xi_+]$ (resp. $\mathbb{E}[\xi_-]$) be the expected cost of subpolicy ξ after action a succeeds (resp. fails).*

Strategy ξ starting with action a_{k+1} and followed optimally has higher expected reward then strategies starting with actions $a_1, \ldots, a_{k+1}$ (and followed optimally) iff

$$\mathbb{E}[\xi_+] > M - \rho + \frac{c_{a_{k+1}}}{p_{a_{k+1}}}$$

and

$$\mathbb{E}[\xi_-] > \frac{M + c_{a_{k+1}} - p_{a_{k+1}}(\mathbb{E}[\xi_+] + \rho)}{1 - p_{a_{k+1}}},$$

where $\rho = \sum_{f:f_a \setminus \phi} r_f$ is the pure immediate reward for successfully performed action a.

The proof of the first inequality:

Proof 3

$$\begin{aligned}
\mathbb{E}[\xi] &= -c_{a_{k+1}} + p_{a_{k+1}}(\mathbb{E}[\xi_+] + \rho) + \bar{p_a}\mathbb{E}[\xi_-] > M \\
-c_{a_{k+1}} + p_{a_{k+1}}(\mathbb{E}[\xi_+] + \rho) &+ p_{\bar{a_{k+1}}}(\mathbb{E}[\xi_+] + \rho - \frac{c_{a_{k+1}}}{p_{a_{k+1}}}) > M \\
\mathbb{E}[\xi_+] &> M - \rho + \frac{c_{a_{k+1}}}{p_{a_{k+1}}}.
\end{aligned}$$

The proof of the second inequality:

Proof 4

$$\begin{aligned}
\mathbb{E}[\xi] &> M \\
-c_a + p_a(\mathbb{E}[\xi_+] + \rho) &+ p_{\bar{a_{k+1}}}\mathbb{E}[\xi_-] > M \\
\mathbb{E}[\xi_-] &> \frac{M + c_{a_{k+1}} - p_{a_{k+1}}(\mathbb{E}[\xi^+] + \rho)}{p_{\bar{a_{k+1}}}}.
\end{aligned}$$

Approximate Solutions for Attack Graph Games with Imperfect Information

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

[1] Department of Computer Science, Agent Technology Center, Czech Technical University in Prague, Prague, Czech Republic
{karel.durkota,viliam.lisy}@agents.fel.cvut.cz
[2] Department of Computing Science, University of Alberta, Edmonton, Canada
[3] Department of Computer Science, Aarhus University, Aarhus, Denmark
bosansky@cs.au.dk
[4] Computer Science Department, University of Texas at El Paso, El Paso, USA
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 [13,29]. Believable HPs, however, are costly

MHR Khouzani et al. (Eds.): GameSec 2015, LNCS 9406, pp. 228–249, 2015.
DOI: 10.1007/978-3-319-25594-1_13

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 [12,19], 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 [24,32], or to calculate security risk measures (e.g., the probability of a successful attack) [14,25]. 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 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 [1,23,26] that once a fact becomes true during an attack, it can never become false again as an effect of any action.

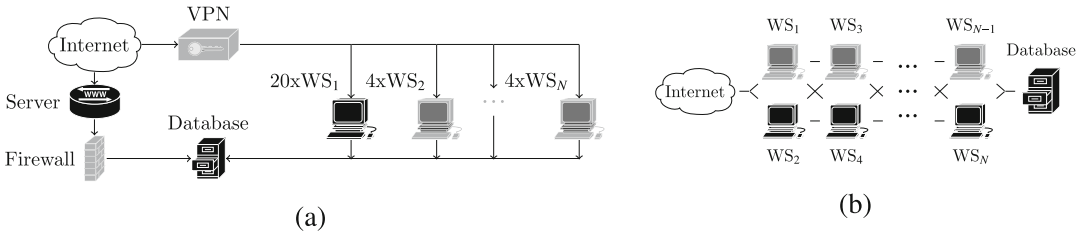


Fig. 1. Simple (a) business-like and (b) chain network topology.

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.

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., [5,21]) 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

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

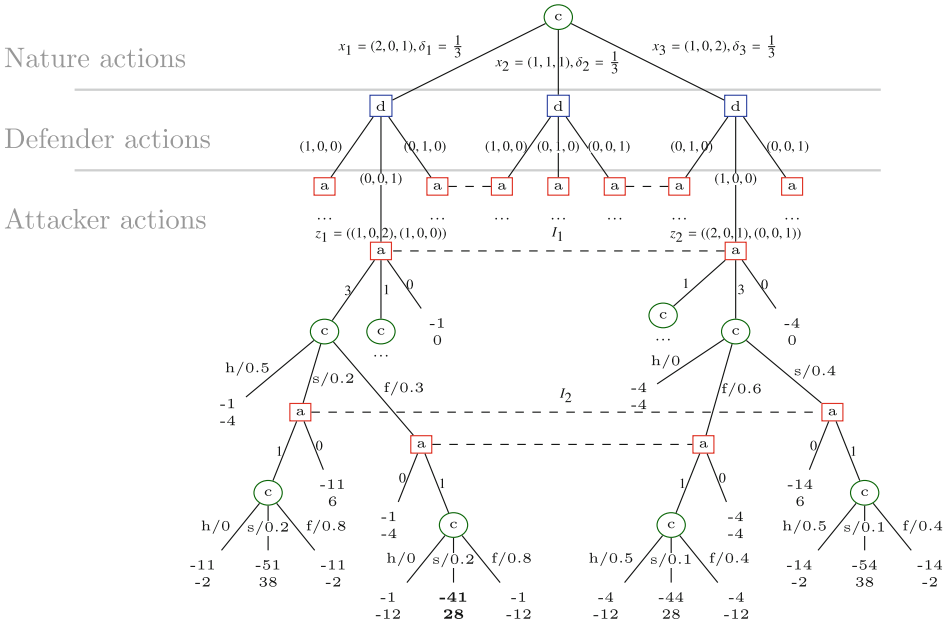


Fig. 2. Simple game tree with $|T| = 3$ host types, basis $b = (1, 0, 1)$, number of hosts $n = 3$ and $k = 1$ HP. The defender's costs for HPs are $c_1^h = 4$ and $c_3^h = 1$. The attacker's attack action 1 (resp. 3) exploits vulnerability of host type 1 (resp. 3), costs $c_1 = 8$ (resp. $c_3 = 4$); 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.

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 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 Sect. 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 (Sect. 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 Sect. 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 Sect. 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 $(\hat{\sigma}_{\mathrm{d}}^{m}, \hat{\pi}_{\mathrm{a}}^{m})$ (if $m = 1$ then $\hat{\sigma}_{\mathrm{d}}^{1}$ 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 $\pi_{\mathrm{a}}^{m} = BR_{\mathrm{a}}(\hat{\sigma}_{\mathrm{d}}^{m})$ in the full game G. If all actions from π_{a}^{m} are included in the restricted game $\hat{G}$, the algorithm returns strategy profile $(\hat{\sigma}_{\mathrm{d}}^{m}, \hat{\pi}_{\mathrm{a}}^{m})$ 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 $\hat{\pi}_{\mathrm{a}}^{m}$ 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 Sect. 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 $\pi_{\mathrm{a}} = BR_{\mathrm{a}}(\sigma_{\mathrm{d}})$ 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 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. Equations 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.

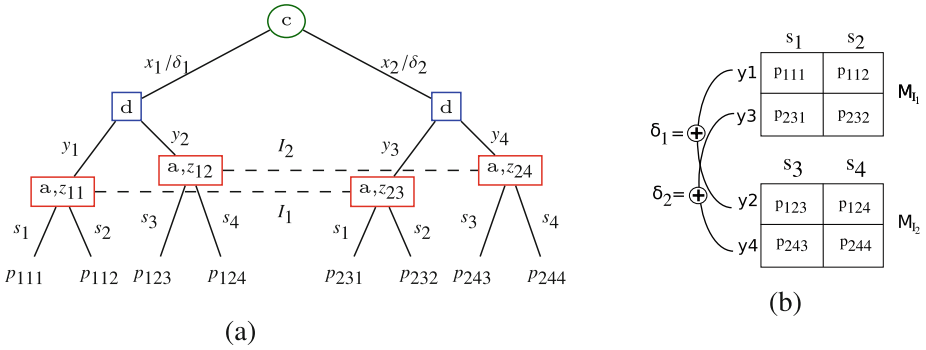


Fig. 3. The extensive-form game in (a) translated into two normal-form games in (b).

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 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) approximation solved with backward induction (Sect. 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) (Sect. 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 Sect. 6.1 we describe networks we use to generate the action graph game, in Sect. 6.2 we discuss an issue of combinatorially large CURB sets for one of the networks, in Sect. 6.3 we analyze the scalability of the approximated models, in Sect. 6.4 we analyze the quality of the strategies found by the approximated models and in Sect. 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 Sect. 6.6 we investigate the defender's regret for imprecisely modeling the attack graph, and conclude with a case-study analysis in Sect. 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 performing 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.

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

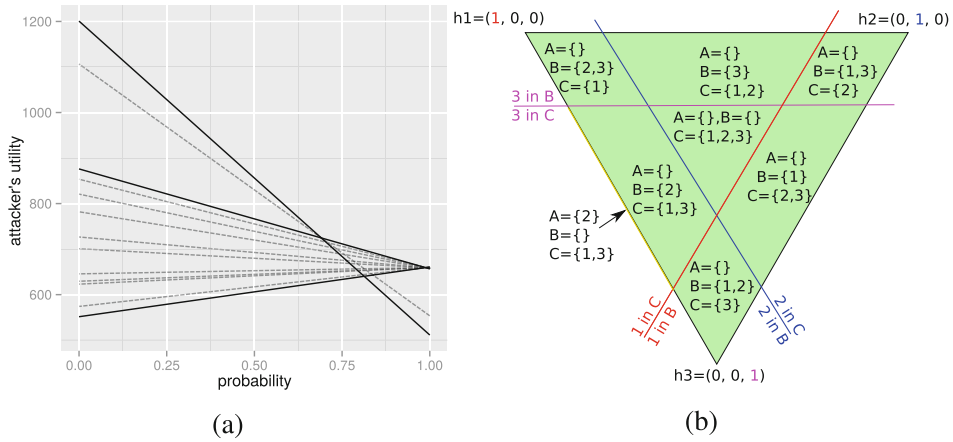
All experiments were run on a 2-core 2.6 GHz processor with 32 GB memory limitation and 2 h of runtime.

6.2 Analytical Approach for CURB for Unstructured Network

The incremental pruning algorithm described in Sect. 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.



(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.

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.

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 h 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.

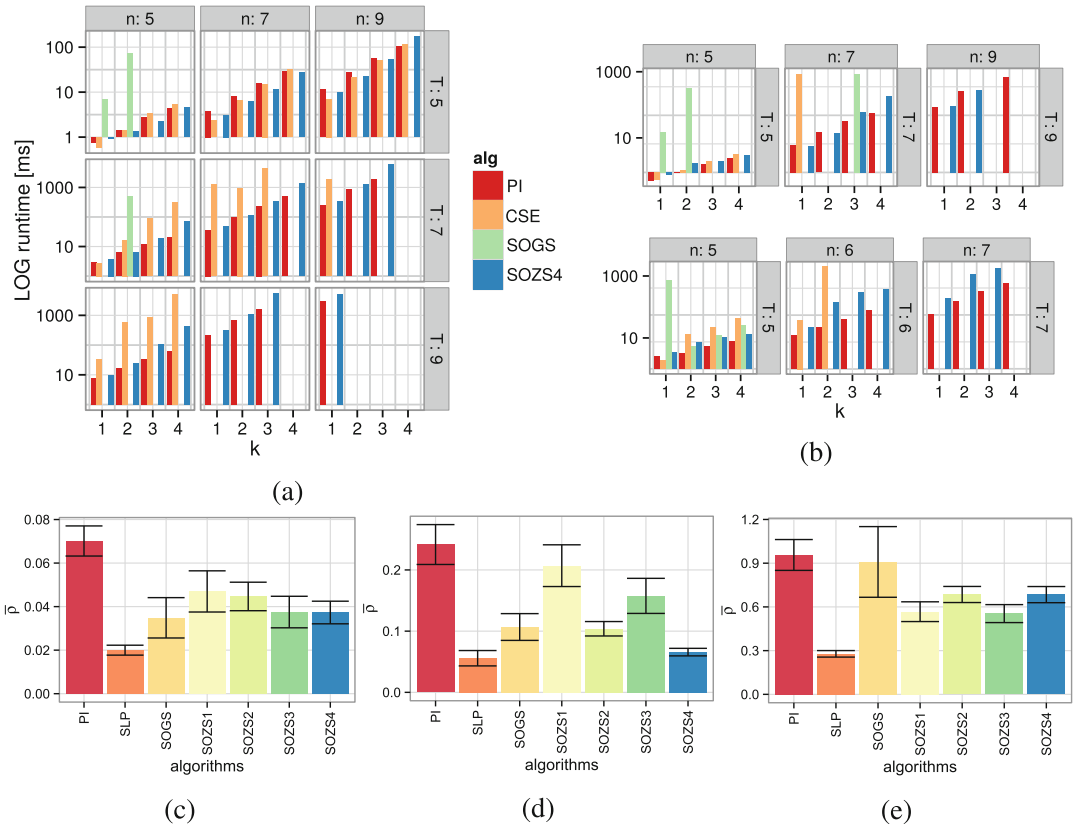


Fig. 5. Comparison of approximations scalability for (a) business, and (b top) chain and (b bottom) unstructured network. In (c), (d) and (e) we compare the defender's upper bound of relative regret of strategies computed with approximation algorithms business, chain and unstructured network, respectively.

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 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 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 $\bar{u} = \frac{1}{|L|} \sum_{l \in L} (|u_{\mathrm{d}}(l) + u_{\mathrm{a}}(l)|)$, 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 $c_t^h \in [0, 0.1r_t]$, 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.

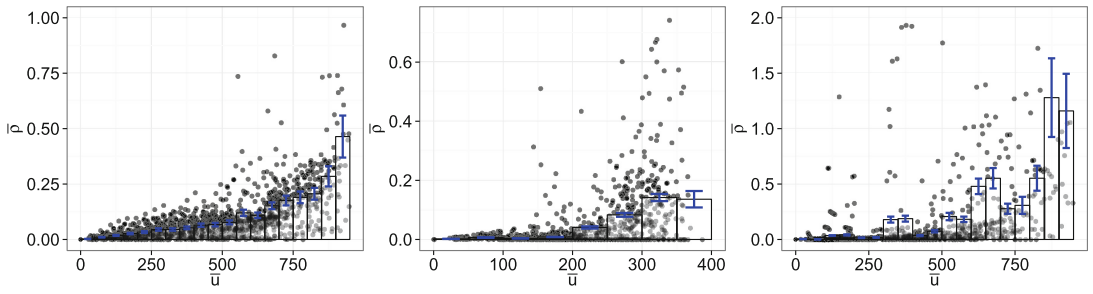


Fig. 6. The defender's relative regret dependence on game *zero-sumness* (computed as average $u_{\mathrm{a}}(l) + u_{\mathrm{d}}(l)$) for (a) business, (b) chain and (c) unstructured networks.

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. Figure 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.

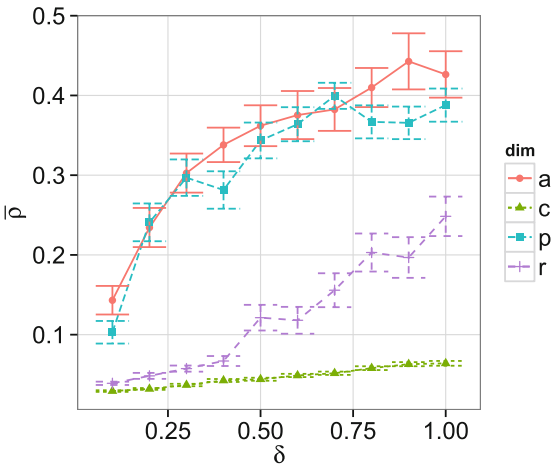


Fig. 7. The defender's utility regret for perturbed action success probabilities, action costs, and host type values.

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,

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

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. 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: Proceedings of CCS, pp. 217–224 (2002)
2. Bacic, E., Froh, M., Henderson, G.: Mulval extensions for dynamic asset protection. Technical report, 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 (2010)
4. Bernheim, B.D.: Rationalizable strategic behavior. Econometrica **52**, 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: Proceedings 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. **51**, 829–866 (2014)
7. Bošanský, B., Čermak, J.: Sequence-form algorithm for computing stackelberg equilibria in extensive-form games. In: Proceedings of AAAI Conference on AI, pp. 805–811 (2015)
8. Carroll, T.E., Grosu, D.: A game theoretic investigation of deception in network security. Secur. Commun. Netw. **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: Proceedings of UAI, pp. 54–61. Morgan Kaufmann Publishers Inc. (1997)
10. Conitzer, V., Korzhyk, D.: Commitment to correlated strategies. In: Proceedings of AAAI, pp. 632–637 (2011)
11. Conitzer, V., Sandholm, T.: Computing the optimal strategy to commit to. In: Proceedings 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: Proceedings 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 (2013)
15. Ingols, K., Lippmann, R., Piwowarski, K.: Practical attack graph generation for network defense. In: Proceedings of ACSAC, pp. 121–130 (2006)
16. Koller, D., Megiddo, N., Von Stengel, B.: Efficient computation of equilibria for extensive two-person games. Games Econ. Behav. **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: Proceedings of ACM EC, pp. 83–92 (2010)
19. Letchford, J., Vorobeychik, Y.: Optimal interdiction of attack plans. In: Proceedings of AAMAS, pp. 199–206 (2013)
20. Littman, M.L.: The witness algorithm: Solving partially observable markov decision processes. Technical report, 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. Secur. Priv. **4**, 85–89 (2006)
23. Noel, S., Jajodia, S.: Managing attack graph complexity through visual hierarchical aggregation. In: Proceedings of ACM VizSEC/DMSEC, pp. 109–118. ACM (2004)
24. Noel, S., Jajodia, S.: Optimal ids sensor placement and alert prioritization using attack graphs. J. Netw. Syst. Manage. **16**, 259–275 (2008)
25. Noel, S., Jajodia, S., Wang, L., Singhal, A.: Measuring security risk of networks using attack graphs. Int. J. Next-Gener. Comput. **1**(1), 135–147 (2010)

26. Ou, X., Boyer, W.F., McQueen, M.A.: A scalable approach to attack graph generation. In: Proceedings of ACM CCS, pp. 336–345. ACM (2006)
27. Ou, X., Govindavajhala, S., Appel, A.W.: Mulval: a logic-based network security analyzer. In: Proceedings of USENIX SSYM. pp. 113–128. USENIX Association, Berkeley (2005)
28. Píbil, R., Lisý, V., Kiekintveld, C., Bošanský, B., Pěchouček, M.: Game theoretic model of strategic honeypot selection in computer networks. In: Grossklags, J., Walrand, J. (eds.) GameSec 2012. LNCS, vol. 7638, pp. 201–220. Springer, Heidelberg (2012)
29. Provos, N.: A virtual honeypot framework. In: Proceedings of USENIX SSYM, pp. 1–14. Berkeley, CA, USA (2004)
30. Qassrawi, M.T., Hongli, Z.: Deception methodology in virtual honeypots. In: Proceedings of NSWCTC, vol. 2, pp. 462–467. IEEE (2010)
31. Sawilla, R.E., Ou, X.: Identifying critical attack assets in dependency attack graphs. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 18–34. Springer, Heidelberg (2008)
32. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: IEEE Symposium Security and Privacy, pp. 273–284. IEEE (2002)
33. Tambe, M.: Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned, 1st edn. Cambridge University Press, New York (2011)
34. Von Stengel, B., Forges, F.: Extensive form correlated equilibrium: definition and computational complexity. Math. Oper. Res. **33**(4), 1002–1022 (2008)

Using Correlated Strategies for Computing Stackelberg Equilibria in Extensive-Form Games

Jiří Čermák[1], **Branislav Bošanský**[1,2], **Karel Durkota**[1], **Viliam Lisý**[1,3], **Christopher Kiekintveld**[4]
[1] Agent Technology Center, Faculty of Electrical Engineering, Czech Technical University in Prague
[2] Department of Computer Science, Aarhus University
[3] Department of Computing Science, University of Alberta
[4] Department of Computer Science, University of Texas at El Paso
jiri.cermak@agents.fel.cvut.cz, branislav.bosansky@agents.fel.cvut.cz, karel.durkota@agents.fel.cvut.cz,
lisy@ualberta.ca, cdkiekintveld@utep.edu

Abstract

Strong Stackelberg Equilibrium (SSE) is a fundamental solution concept in game theory in which one player commits to a strategy, while the other player observes this commitment and plays a best response. We present a new algorithm for computing SSE for two-player extensive-form general-sum games with imperfect information (EFGs) where computing SSE is an NP-hard problem. Our algorithm is based on a correlated version of SSE, known as Stackelberg Extensive-Form Correlated Equilibrium (SEFCE). Our contribution is therefore twofold: (1) we give the first linear program for computing SEFCE in EFGs without chance, (2) we repeatedly solve and modify this linear program in a systematic search until we arrive to SSE. Our new algorithm outperforms the best previous algorithms by several orders of magnitude.

Introduction

The roles of players in many games are often asymmetric. One example is the ability of one player (*the leader*) to commit to a strategy, to which the other players (*the followers*) react by playing their best response. In real-world scenarios, the leader can model a market leader with the power to set the price for items or services, or a defense agency committing to a security protocol to protect critical facilities. Optimal strategies for the players in such situations are described by the Strong Stackelberg Equilibrium (SSE) (Leitmann 1978; von Stengel and Zamir 2004). There are many examples of successful applications of SSE in infrastructure protection (Tambe 2011) as well as protecting the environment and wildlife (Fang, Stone, and Tambe 2015).

In most of the existing works, the game models are simplified and do not consider the sequential interaction among players (Pita et al. 2008; Tsai et al. 2009; Shieh et al. 2012; Jiang et al. 2013). One reason is computational complexity, since computing SSE is often NP-hard when sequential interaction is allowed (Letchford and Conitzer 2010). Another reason is the lack of practical algorithms. The only algorithm designed specifically for computing SSE in two-player imperfect-information general-sum extensive-form games (EFGs) was only introduced recently (Bosansky and Cermak 2015) and formulates the problem as a mixed-integer variant of sequence-form linear program (referred to as BC15). However, the scalability of BC15 is limited as it contains a binary variable for each sequence of actions of the follower.

Our main contribution is a novel algorithm computing SSE in EFGs that offers a dramatic speed-up in computation time compared to BC15. The key idea behind our algorithm is in computing a variant of SSE, where the leader commits to correlated strategies – i.e., the leader can send signals to the follower, and the best response of the follower is to follow these signals. We use this variant to find the original SSE by systematically restricting which signals the leader can send to the follower. This variant of SSE has previously been studied in single step games (Conitzer and Korzhyk 2011), finite turn-based and concurrent-move games (Bosansky et al. 2015), infinite concurrent-move stochastic games (Letchford et al. 2012), and security games (Xu et al. 2015; Rabinovich et al. 2015; Durkota et al. 2015). Formally, it has been defined as Stackelberg Extensive-Form Correlated Equilibrium (SEFCE) in (Bosansky et al. 2015). While it was shown that the utility value for the leader in SSE cannot be closely approximated by SEFCE (Letchford et al. 2012), we show that one can use SEFCE to compute SSE. Since there was no previously known algorithm for computing SEFCE in EFGs, we also show that SEFCE can be found in polynomial time in EFGs without chance.

The paper is structured as follows. After introducing the formalism of EFGs, we formally define both SSE and SEFCE, and give an example of a game where these concepts differ. Next, we present the linear program (LP) for computing SEFCE in EFGs without chance (we describe a modified LP for EFGs with chance in the full version of the paper). Afterwards, we present our algorithm for computing SSE in EFGs that iteratively solves the LP for SEFCE with additional constraints until SSE is reached. Finally, we provide three variants of our algorithm and show that each variant significantly improves the computation time compared to BC15 on randomly generated games and an example of a search game.

Technical Background

Extensive-form games model sequential interactions between players and can be visually represented as game trees. Formally, a two-player EFG is defined as a tuple $G = (\mathcal{N}, \mathcal{H}, \mathcal{Z}, \mathcal{A}, u, \mathcal{C}, \mathcal{I})$: $\mathcal{N} = \{l, f\}$ is a set of players, the leader and the follower. We use i to refer to one of the players, and $-i$ to refer to his opponent. $\mathcal{H}$ denotes a finite set of *nodes* in the game tree. Each node corresponds to a unique *history* of actions taken by all players and chance from the root of the game; hence, we use the terms history and node interchangeably. We say that h is a *prefix* of h' ($h \sqsubseteq h'$) if h lies on a path from the root of the game tree to h'. $\mathcal{A}$ denotes the set of all actions. $\mathcal{Z} \subseteq \mathcal{H}$ is the set of all *terminal nodes* of the game. For each $z \in \mathcal{Z}$ we define a *utility function* for each player i ($u_i : \mathcal{Z} \to \mathbb{R}$). Chance player selects actions based on a fixed probability distribution known to all players. Function $\mathcal{C} : \mathcal{H} \to [0, 1]$ denotes the probability of reaching node h due to chance; $\mathcal{C}(h)$ is the product of chance probabilities of all actions in history h.

Imperfect observation of player i is modeled via *information sets* $\mathcal{I}_i$ that form a partition over $h \in \mathcal{H}$ where i takes action. Player i cannot distinguish between nodes in any information set $I \in \mathcal{I}_i$. We overload the notation and use $A(I_i)$ to denote possible actions available in each node from information set I_i. We assume that action a uniquely identifies the information set where it is available. We assume *perfect recall*, which means that players remember history of their own actions and all information gained during the course of the game. As a consequence, all nodes in any information set I_i have the same history of actions for player i.

Pure strategies Π_i assign one action for each $I \in \mathcal{I}_i$. A more efficient representation in the form of *reduced pure strategies* Π_i^* assigns one action for each $I \in \mathcal{I}_i$ reachable while playing according to this strategy. A *mixed strategy* $\delta_i \in \Delta_i$ is a probability distribution over Π_i. For any pair of strategies $\delta \in \Delta = (\Delta_l, \Delta_f)$ we use $u_i(\delta) = u_i(\delta_i, \delta_{-i})$ for the expected outcome of the game for player i when players follow strategies δ. A *best response* of player i to the opponent's strategy δ_{-i} is a strategy $\delta_i^{BR} \in BR_i(\delta_{-i})$, where $u_i(\delta_i^{BR}, \delta_{-i}) \geq u_i(\delta_i', \delta_{-i})$ for all $\delta_i' \in \Delta_i$.

Strategies in EFGs with perfect recall can be compactly represented by using the sequence form (Koller, Megiddo, and von Stengel 1996). A *sequence* $\sigma_i \in \Sigma_i$ is an ordered list of actions taken by a single player i in history h. $\emptyset$ stands for the empty sequence (i.e., a sequence with no actions). A sequence $\sigma_i \in \Sigma_i$ can be extended by a single valid action a taken by player i, written as $\sigma_i a = \sigma_i'$. We say that σ_i is a *prefix* of σ_i' ($\sigma_i \sqsubseteq \sigma_i'$) if σ_i' is obtained by finite number (possibly zero) of extensions of σ_i. We use $\sigma_i(I_i)$ and $\sigma_i(h)$ to denote the sequence of i leading to I_i and h, respectively. We use the function $I_i(\sigma_i')$ to obtain the information set in which the last action of the sequence σ_i' is taken. For an empty sequence, function $I_i(\emptyset)$ returns the information set of the root node. A mixed strategy of a player can now be represented as a *realization plan* ($r_i : \Sigma_i \to \mathbb{R}$). A realization plan for a sequence σ_i is the probability that player i will play σ_i under the assumption that the opponent plays to allow the actions specified in σ_i to be played. By $g_i : \Sigma_l \times \Sigma_f \to \mathbb{R}$ we denote the *extended utility function*, $g_i(\sigma_l, \sigma_f) = \sum_{z \in \mathcal{Z} | \sigma_l(z) = \sigma_l \wedge \sigma_f(z) = \sigma_f} u_i(z) C(z)$. If no leaf is reachable with pair of sequences σ, value of g_i is 0.

Solution Concepts in EFGs

Here we provide a formal definition of Strong Stackleberg Equilibrium (SSE) (e.g., in (Leitmann 1978)) and Stackelberg Extensive-Form Correlated Equilibrium (SEFCE) (Bosansky et al. 2015) and give the intuition on an example game.

Definition 1. *A strategy profile $\delta = (\delta_l, \delta_f)$ is a* Strong Stackelberg Equilibrium *if δ_l is an optimal strategy of the leader given that the follower best-responds. Formally:*

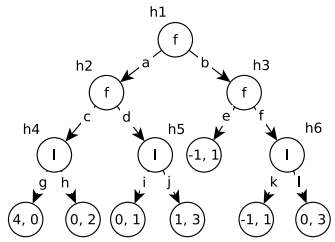
$$(\delta_l, \delta_f) = \underset{\delta_l' \in \Delta_l, \delta_f' \in BR_i(\delta_l')}{\arg\max} u_l(\delta_l', \delta_f'). \tag{1}$$

The SSE of the game in Figure 1 (the first utility in every leaf is for the leader, second for the follower) prescribes the leader to commit to playing g in h_4, j in h_5, and k in h_6. The strategy of the follower is then to play a in h_1 and d in h_2, leading to the expected utility of 1 for the leader.

In SEFCE we allow the leader to send signals to the follower and condition his strategy on sent signals. More specifically, the leader chooses $\pi_f^* \in \Pi_f^*$ as the recommendations for the follower according to SEFCE before the game starts. The actual recommendation to play some action $a \in A(I_f)$ is revealed to the follower only after he reaches I_f. Therefore, the follower only knows the past and current recommendations, and the probability distribution from which the recommendations are drawn in the future.

Definition 2. *A probability distribution λ on reduced pure strategy profiles Π^* is called a* Stackelberg Extensive-Form Correlated Equilibrium *if it maximizes the leader's utility subject to the constraint that whenever play reaches an information set I where the follower can act, the follower is recommended an action a according to λ such that the follower cannot gain by unilaterally deviating from a in I and possibly in all succeeding information sets given the posterior on the probability distribution of the strategy of the leader, defined by the actions taken by the leader so far.*

The middle table in Figure 1 represents the distribution λ forming the SEFCE of the example game (rows are labeled by Π_l^*, columns by Π_f^*). The leader chooses the signals to the follower to be either $\{a, c\}$ or $\{a, d\}$ based on the probability distribution depicted in the table. Afterwards, the corresponding column defines a valid mixed strategy for the leader and the signals the follower receives in his information sets. More specifically, the follower can receive either c or d in h_2. When the follower receives the recommendation to play c, the leader commits to mix uniformly between g and h in h_4 and to play i in h_5. When the follower receives d as the recommendation, the leader commits to playing g in h_4 and j in h_5. Finally in h_1 the follower is recommended to play a, while the leader commits to play k in h_6 to ensure that the follower does not deviate from playing a. The expected utility of the leader is 1.5 in SEFCE. Note that SSE in this representation always corresponds to a table where only a single column of the follower has non-zero values.



	$\{a,c\}$	$\{a,d\}$	$\{b,e\}$	$\{b,f\}$
$\{g,i,k\}$	0.25	0	0	0
$\{g,i,l\}$	0	0	0	0
$\{g,j,k\}$	0	0.5	0	0
$\{g,j,l\}$	0	0	0	0
$\{h,i,k\}$	0.25	0	0	0
$\{h,i,l\}$	0	0	0	0
$\{h,j,k\}$	0	0	0	0
$\{h,j,l\}$	0	0	0	0

	$\emptyset$	(a)	(b)	(a,c)	(a,d)	(b,e)	(b,f)
$\emptyset$	1	1	0	0.5	0.5	0	0
(g)	0.75	0.75	0	0.25	0.5	-	-
(h)	0.25	0.25	0	0.25	0	-	-
(i)	0.5	0.5	0	0.5	0	-	-
(j)	0.5	0.5	0	0	0.5	-	-
(k)	1	1	0	-	-	0	0
(l)	0	0	0	-	-	0	0

Figure 1: (Left) EFG with different SEFCE and SSE. (Middle) SEFCE distribution over Π^*. (Right) SEFCE correlation plan.

LP for Computing SEFCE

To compactly represent the behavior described in the middle table in Figure 1, we use a *correlation plan* (von Stengel and Forges 2008) that is quadratic in the size of the game tree, instead of the exponential representation using Π^*. A correlation plan for a sequence pair $p(\sigma_l, \sigma_f)$ represents the expected probability that σ_l will be played if actions from σ_f are recommended to the follower. In order to model SEFCE strategies using the correlation plan, the follower must be able to determine whether following the signal is the best response. Therefore, we need to specify the plan for so called *relevant sequences*. Consider our game in Figure 1; when the follower receives, for example, signal c in h_2, the follower needs to know the commitment of the leader in the related part of the tree – i.e, in both nodes h_4, h_5 – to evaluate whether following the signal is the best response.

Definition 3. *A pair of sequences (σ_l, σ_f) is termed* relevant *if and only if either $\sigma_f = \emptyset$, or $\exists h, h' \in \mathcal{H}, h' \sqsubseteq h; \sigma_l = \sigma_l(h) \wedge h' \in I_f(\sigma_f)$.*

By $rel(\sigma_i)$ we denote the set of sequences of $-i$ which form a relevant pair with σ_i. In our example $rel((a)) = rel((b)) = \Sigma_l$, $rel((b,e)) = rel((b,f)) = \{\emptyset, (k), (l)\}$, and $rel((a,c)) = rel((a,d)) = \{\emptyset, (g), (h), (i), (j)\}$.

Definition 4. *A* correlation plan *(von Stengel and Forges 2008) is a partial function $p : \Sigma_l \times \Sigma_f \to \mathbb{R}$ such that there is a probability distribution λ on the set of reduced strategy profiles Π^* so that for each relevant sequence pair (σ_l, σ_f), the term $p(\sigma_l, \sigma_f)$ is defined and fulfills $p(\sigma_l, \sigma_f) = \sum_{(\pi_l, \pi_f) \in \Pi^*} \lambda(\pi_l, \pi_f)$ where π_l, π_f prescribe playing all of the actions in σ_l and σ_f, respectively.*

Let us now describe the SEFCE strategies (middle table in Figure 1) using the correlation plan (the right table; rows are labeled by Σ_l, columns by Σ_f). Every column of the table corresponds to an expected probability of the occurrence of the leader's sequences when the follower follows his recommendations in the future, and gets the recommendation to play the σ_f corresponding to this column. We use '-' to mark irrelevant sequence pairs. The entry for every σ_l, σ_f is the sum of all the entries corresponding to the pure strategies containing all the actions from σ_l and σ_f in the middle table of Figure 1. The behavior in columns corresponding to sequences (a,c) and (a,d) matches the behavior discussed in the previous section. The behavior in columns for sequence $\emptyset$ and (a) corresponds to the expected probability of playing sequences of the leader given the probability of recommendations, e.g., the probability of playing g for the recommendation (a) is equal to $p((g),(a,c)) + p((g),(a,d))$ (in this case the leader will play uniformly either g with probability 0.5 according to the column for (a,c) or g with probability 1 according to the column for (a,d)). Probabilities in column for (a) allow the follower to evaluate his choices in h_1.

Now we are ready to describe the LP for computing SEFCE in EFGs without chance that uses correlation plan:

$$\max_{p,v} \sum_{\sigma_l \in \Sigma_l} \sum_{\sigma_f \in \Sigma_f} p(\sigma_l, \sigma_f) g_l(\sigma_l, \sigma_f) \tag{2}$$

$$\text{s.t.} \quad p(\emptyset, \emptyset) = 1; \quad 0 \le p(\sigma_l, \sigma_f) \le 1 \tag{3}$$

$$p(\sigma_l(I), \sigma_f) = \sum_{a \in A(I)} p(\sigma_l(I)a, \sigma_f) \;\; \forall I \in \mathcal{I}_l, \forall \sigma_f \in rel(\sigma_l) \tag{4}$$

$$p(\sigma_l, \sigma_f(I)) = \sum_{a \in A(I)} p(\sigma_l, \sigma_f(I)a) \;\; \forall I \in \mathcal{I}_f, \forall \sigma_l \in rel(\sigma_f) \tag{5}$$

$$v(\sigma_f) = \sum_{\sigma_l \in rel(\sigma_f)} p(\sigma_l, \sigma_f) g_f(\sigma_l, \sigma_f) + \\ + \sum_{I \in \mathcal{I}_f;\, \sigma_f(I) = \sigma_f} \sum_{a \in A_f(I)} v(\sigma_f a) \;\; \forall \sigma_f \in \Sigma_f \tag{6}$$

$$v(I, \sigma_f) \ge \sum_{\sigma_l \in rel(\sigma_f)} p(\sigma_l, \sigma_f) g_f(\sigma_l, \sigma_f(I)a) \; + \sum_{I' \in \mathcal{I}_f;\, \sigma_f(I') = \sigma_f(I)a} v(I', \sigma_f) \\ \forall I \in \mathcal{I}_f, \forall \sigma_f \in \bigcup_{h \in I} rel(\sigma_l(h)), \forall a \in A(I) \tag{7}$$

$$v(\sigma_f(I)a) = v(I, \sigma_f(I)a) \qquad \forall I \in \mathcal{I}_f, \forall a \in A(I) \tag{8}$$

The LP is derived from the computation of Extensive-Form Correlated Equilibrium (von Stengel and Forges 2008) by omitting the incentive constraints for the leader and maximizing the expected utility of the leader (this is similar to the approach in normal-form games (Conitzer and Korzhyk 2011)). Constraints (3) to (5) ensure that p is a well-formed correlation plan. Constraint (6) ensures that $v(\sigma_f)$ represents the expected utility of playing σ_f for the follower, when he follows his recommendations. The first sum represents the expected utility of the leaves reached by playing according to σ_l and σ_f, the second sum represents the contribution of the expected utility from information sets reached by the continuations of σ_f. Constraint (7) ensures that $v(I_f, \sigma_f)$ is the maximum over all possible sequences leaving I_f (denoted as $\sigma_f(I_f)a$ for all $a \in A(I_f)$) after the follower has received recommendation σ_f. Finally, constraint (8) forces the move recommended to the follower in I_f to be optimal.

Definition 5. *We say that p uses* inconsistent recommendation *in $I \in \mathcal{I}_f$ if and only if p defines two different recommendations for the follower in I. Formally, $\exists a, a' \in A(I), a \neq a', \exists \sigma_l, \sigma'_l \in \bigcup_{h \in I} \sigma_l(h)$ $p(\sigma_l, \sigma_f(I)a) > 0 \wedge p(\sigma'_l, \sigma_f(I)a') > 0$. If there exists no such information set we say that p uses only* consistent recommendations.

Theorem 1. *Assume a solution of the LP as described in eqs.* (2) *to* (8) *such that there are only consistent recommendations for the follower. There is a SSE strategy that can be found in polynomial time from the p variables.*

Proof. First, we show how the strategy of the leader is constructed. In every $I \in \mathcal{I}_l$ there is a subset Σ_r of relevant sequences $rel(\sigma_l(I))$ played with a positive probability. The behavior in I is specified by $p(\sigma_l(I)a, \sigma_f)$ for all $a \in A(I)$ for arbitrary $\sigma_f \in \Sigma_r$, as the behavior is the same for all $\sigma'_f \in \Sigma_r$. This is guaranteed by constraint (5) – it forces the probability of $p(\sigma_l, \sigma'_f)$ to be equal to the sum of $p(\sigma_l, \sigma''_f)$ over all extensions σ''_f of σ'_f. Since there can be only a single extension played with positive probability (assumed consistent recommendations) the probabilities must be equal.

For every $I \in \mathcal{I}_f$ there exists at most one action $a \in A(I)$ with $p(\sigma_l, \sigma_f a) > 0$ for some $\sigma_l \in \Sigma_l$ and $\sigma_f = \sigma_f(I)$ (consistent recommendations). By taking these actions and arbitrary actions in information sets where there is no such action a, we obtain a pure strategy for the follower. Finally, due to the correctness of the strategy of the leader proved in the previous step and constraints (6–8), this pure strategy is a best response of the follower. □

Theorem 2. *Assume a solution of the LP as described in eqs.* (2) *to* (8)*. The objective value is greater than or equal to the expected utility of the leader in SSE.*

Proof. Theorem 1 shows that in case the leader uses only consistent recommendations, the value of the LP corresponds to the expected utility of the leader in SSE. If the leader can also use inconsistent recommendations, the value of the LP can be only greater or equal. □

Algorithm Computing SSE

In this section we describe the algorithm for computing SSE. The algorithm uses the linear program for computing SEFCE in case the game is without chance. Otherwise, a slightly modified version of this LP is required, however, the algorithm remains the same. Due to the space constraints, we describe this modified LP in details in the full version of the paper and refer to one of these two LPs as UB-SSE-LP.

The high level idea of our algorithm (depicted in Algorithm 1) is a recursive application of the following steps: (1) solve the UB-SSE-LP, (2) detect the set of information sets of the follower with inconsistent recommendations $\mathcal{I}_{in}$, (3) restrict the leader to use only consistent recommendations in $\mathcal{I}_{in}$ by adding new constraints to the UB-SSE-LP. Restrictions are added cumulatively, until we arrive at a restricted UB-SSE-LP yielding only consistent p. The expected utility for the leader and the correlation plan p in this solution correspond to a candidate for the expected utility of the leader and the strategies of players in SSE. It is only a solution candidate, since we have enforced actions, which may not be a part of SSE.

Input: An UB-SSE-LP P
Output: leader's expected utility and strategy profile in SSE

```
1  M ← {(∞, ∅)}; LB ← −∞; p_c ← ∅
2  while M ≠ ∅ do
3      (UB, m) ← max(M)
4      if UB < LB then
5          return (LB, p_c)
6      apply(m, P)
7      if feasible(P) then
8          (value, p) ← solve(P)
9          I_in ← inconsistentRecommendations(p)
10         if I_in = ∅ then
11             if value > LB then LB ← value; p_c ← p
12         else addRestrictions((UB, m), M, I_in, value)
13     revert(m, P)
14 return (LB, p_c)
```

Algorithm 1: Algorithm for computing the SSE.

In more details, Algorithm 1 assumes as the input the UB-SSE-LP P for the game we want to solve. By *modification* $mod = (UB, m)$ we denote a pair of constraints m, to be added to P, and the upper bound UB on the value of P after adding the constraints. M is the set of modifications to be explored during the search for the SSE sorted in descending order of UB. The variable LB stores the expected utility for the leader in the best solution candidate found so far p_c. The main cycle of the algorithm starts on line 2, we iterate until there are no possible modifications of P left. On line 3 we remove the modification with the highest UB from M. We choose such $mod = (UB, m)$ in order to first explore modifications with the potential to lead to solution candidates with the highest expected utility for the leader. The algorithm verifies whether the modification mod (the one with the highest UB value) can improve the current best solution candidate (line 4). If not, the algorithm terminates and the best candidate found so far is the SSE. Otherwise, we add the constraints in m to P (line 6). If the modified P is feasible, the algorithm solves the LP (line 8) obtaining the expected utility of the leader and the correlation plan p. We find the set of information sets where the follower gets an inconsistent recommendation in p (line 9). If p uses only consistent recommendations, this solution corresponds to a solution candidate. If the expected utility for the leader is higher than for the best solution candidate found so far, we replace it (line 11). If $\mathcal{I}_{in}$ is not empty, we generate new modifications to be applied to P and add them to M (line 12). The function $addRestrictions$ will be discussed in more detail in the next subsection, as we explored several options of the modification generation. Finally, we revert the changes in m (line 13). If there are modifications left to be explored in M we continue with the next iteration. Every modification contains all the constraints added to the original P given as an input, therefore after $revert$ we again obtain the original P.

When we enforce the whole strategy of the follower to be consistent with the SSE, the solution of the LP corresponds to the SSE, as it now maximizes the expected utility of the leader under the restriction the follower gets recom-

```
1  addRestrictions((UB, m), M, I_in, value)
2  |  I ← getShallowest(I_in)
3  |  for a ∈ A(I) do
4  |  |  UB_a ← value; m_a ← m
5  |  |  for σ_l ∈ rel(σ_f(I)a) do
6  |  |  |  m_a ← m_a ∪ {p(σ_l, σ_f(I)) = p(σ_l, σ_f(I)a)}
7  |  |  M ← M ∪ {(UB_a, m_a)}
```

Algorithm 2: SI-LP.

```
1  addRestrictions((UB, m), M, I_in, value)
2  |  I_c ← chooseSets(I_in)
3  |  UB' ← ∞; m' ← m
4  |  for I ∈ I_c do
5  |  |  A_c ← {a ∈ A(I) | ∃σ_l ∈ Σ_l p(σ_l, σ_f(I)a) > 0}
6  |  |  for a ∈ A_c do
7  |  |  |  for σ_l ∈ rel(σ_f(I)a) do
8  |  |  |  |  m' ← m' ∪ {p(σ_l, σ_f(I)a) ≤ b_a}
9  |  |  |  m' ← m' ∪ {Σ_{a∈A_c} b_a = 1}
10 |  |  M ← M ∪ {(UB', m')}
```

Algorithm 3: MILP.

mended the pure best response to the strategy of the leader. It remains to be shown, that we are guaranteed to add modifications to M which force the correct set of actions of the follower in every version of $addRestrictions$. The final correctness arguments will be provided after discussing the $addRestrictions$ versions used.

Rules for Restricting Follower's Behavior

We examine different approaches for method $addRestrictions$ that generates new modifications of UB-SSE-LP resulting in three different variants of our new algorithm.

In Algorithm 2 we describe the first version of the function $addRestrictions$, labeled SI-LP. On line 2 we find the shallowest information set I, where the follower receives an inconsistent recommendation. We generate modification for every $a \in A(I)$. Every such modification enforces corresponding $a \in A(I)$ to be recommended deterministically. The upper bound is set to the value of P computed before invoking $addRestrictions$. The shallowest information set is chosen to avoid unnecessary modifications in deeper parts of the game tree, which might end up not being visited at all. This version of $addRestrictions$ transforms Algorithm 1 to branch and bound algorithm.

By adding $mod = (UB, m)$ to M for every action in the shallowest information set with inconsistent recommendations, until no such set exists, we ensure that all of the actions which might form a part of SSE will be tried. The behavior in information sets with consistent recommendation need not be restricted, as we are sure that the follower has no incentive to deviate and therefore plays his best response maximizing the expected utility of the leader. Finally, since we assign to UB a value which forms an upper bound on the solution of P after adding constraints m, we are sure that if we terminate the algorithm on line 4 in Algorithm 1, there is indeed no possibility to encounter a solution candidate better than the one yielding the current LB.

A second option, presented in Algorithm 3, chooses $\mathcal{I}_c \subseteq \mathcal{I}_{in}$ (line 2) and restricts the recommendations in every $I \in \mathcal{I}_c$. The restriction in I is done in the following way. First, detect the subset A_c of $A(I)$ of actions which are recommended with positive probability (line 5) and make the recommendation mutually exclusive using binary variables (lines 8 and 9), converting the LP P to a mixed integer linear program (MILP). We use two options of creating $\mathcal{I}_c$. First, we create a singleton containing only the shallowest $I \in \mathcal{I}_{in}$, we refer to this algorithm as SI-MILP. Second, we let $\mathcal{I}_c = \mathcal{I}_{in}$, we refer to this algorithm as AI-MILP. Algorithm 1 using both SI-MILP and AI-MILP closely resembles constraint generation, with the difference that additional binary variables are also added in every iteration.

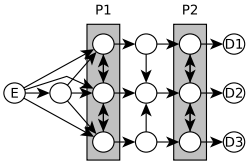
If we introduce a binary variable for every action of the follower in the game, we are guaranteed to obtain the SSE, as the MILP then finds a strategy profile maximizing the expected utility of the leader (ensured by the objective), while the follower plays a pure best response to the strategy of the leader (breaking ties in favor of the leader due to the objective), which is the definition of the SSE. If we create some partial enforcement of consistent recommendations using the binary variables and we obtain a pure strategy for the follower then this is again SSE, since the enforcement in the rest of the game would not make any difference as the follower already gets consistent recommendations there. Finally, since we restrict the follower's recommendations until consistent recommendations are obtained, both MILP based rules indeed guarantee to find the SSE.

Experimental Evaluation

We now turn to the experimental evaluation of the three described variants of our algorithm for computing an SSE. We use BC15 (Bosansky and Cermak 2015) as a baseline, state-of-the-art algorithm for computing SSE in EFGs. Single-threaded IBM CPLEX 12.5 solver was used to compute all the (MI)LPs. We use two different domains previously used for evaluation of BC15: a search game representing the scenario where security units defend several targets against an attacker, and randomly generated games.

Search Game. The search game is played on a directed graph (see Figure 2). The follower aims to reach one of the destination nodes (D1 – D3) from starting node (E) in a given number of moves, while the leader aims to catch the follower with one of the two units operating in the shaded areas of the graph (P1 and P2). The follower receives different reward for reaching different destination node (the reward is randomly selected from the interval $[1, 2]$). The leader receives positive reward 1 for capturing the follower. Once the follower runs out of moves without reaching any goal or being captured, both players receive 0. The follower leaves tracks in the visited nodes that can be discovered if the leader visits the node. The follower can erase the tracks in the current node (it takes one turn of the game). The follower does not know the position of the patrolling units, the leader observes only the tracks left by the follower.

Randomly Generated Games. We use randomly generated games, where in each state of the game the number of



Algs \ Steps	4	5	6
BC15	25	863	42,261
SI-LP	9	212	10,790
AI-MILP	6	257	11,997
SI-MILP	5	272	11,263

Figure 2: (Left) Search game graph. (Right) Runtimes in seconds for the search game with increasing depth.

Table 1: Number of games solved in given time intervals.

Algs\Runtime	1s	5s	30s	2min	25min	4h
BC15	2	12	137	245	393	55
SI-LP	583	191	54	12	4	0
AI-MILP	529	259	51	5	0	0
SI-MILP	483	279	72	9	1	0

available actions is randomly generated up to a given parameter $\{2, \ldots, \max_A\}$. Each action leads to a state where the opponent is to move and also generates *an observation* for the opponent. An observation is a number from a set $\{1, \ldots, \max_O\}$ and determines partitioning of the nodes into the information sets – for player i, the nodes h with the same history of moves $\sigma_i(h)$ and the observations generated by the actions of the opponent $-i$ belong to the same information set. We generate games of differing sizes by varying parameters $\max_A = \{3, 4\}$, $\max_O = \{2, 3\}$, and depth of the game (up to 5 actions for each player). The utility for the players is randomly generated in the interval $[-100, 100]$. The utilities are correlated with factor set to -0.5 (1 represents identical utilities, -1 zero-sum utilities).

Results

The runtime results on random games are depicted in the top graph of Figure 3. The x-axis shows the number of realization plans of the follower, while the y-axis depicts the time in seconds needed to solve a given instance (both axes are logarithmic). The number of realization plans of the follower is a good estimate of how difficult the game is to solve as it reflects both the size of the game as well as the structure of information sets. Each point represents the mean time needed for every algorithm to solve the instances from a given time interval (at least 600 different instances). The standard errors of the mean values are very small compared to the differences between algorithms and not visible in the graph.

The results show that each of the variants significantly outperforms the previous state-of-the-art algorithm BC15. It typically takes around 10 minutes for BC15 to solve games with 10^6 realization plans of the follower, while our algorithms were often able to find solutions under a second. AI-MILP performs best on average, since it is the least sensitive to the different structure of the instances solved. AI-MILP fixes the behavior in a higher number of information sets and so it preforms a successful trade off between the complexity of solving a single MILP and the number of MILP invocations. The second best approach on average is the SI-MILP. The average performance is slightly worse due to a higher number of MILP invocations needed to solve more difficult

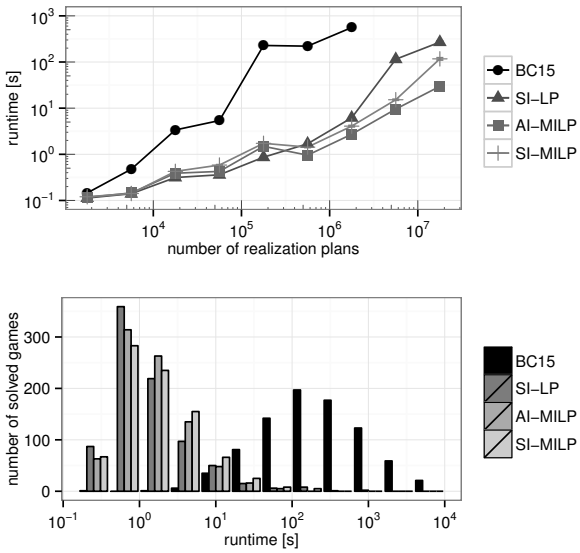


Figure 3: (Top) Runtimes on randomly generated games. (Bottom) Number of solved games in given time intervals.

instances (i.e., the ones with many inconsistent recommendations). Finally, the SI-LP has the worst average performance out of the variants of our new algorithm, since SI-LP needs even more LP invocations on more difficult instances in comparison to the previous variants of our algorithm.

Additionally, we provide in the bottom graph of Figure 3 a histogram for number of instances solved (y-axis) within a time interval (x-axis). The results were calculated on random games with the number of realization plans from interval $[3 \cdot 10^5, 3 \cdot 10^6]$. Despite a slightly worse averge performance of SI-LP on these instances, it solved the highest number of instances very fast compared to the other two variants (SI-LP solves 69% of the instances under 1 second, while AI-MILP solves 62% and SI-MILP 57%). The histogram also shows the reason behind the worse average performance of SI-LP. There are multiple instances that SI-LP solves in more then 200 seconds, while such outliers are not present for the latter two variants (the worst outlier for SI-LP took 773 seconds, while the longest time for SI-MILP was 146 seconds, for AI-MILP 57 seconds and for BC15 2.5 hours). For clarity we provide the same data in coarser intervals in Table 1, where the outliers are clearly visible (the label of column represents the upper bound of the corresponding time interval, the label of the column to the left the lower bound of the time interval). The results show that SI-LP is more efficient on instances where the advantage in using the correlated strategies is marginal and there are only few information sets with inconsistent recommendations. On the other hand, AI-MILP offers higher robustness across different instances.

Finally, in Figure 2 we present the results on the search game. All the new approaches performed similarly, outperforming the BC15 in every setting. This shows that our algorithm outperforms BC15 even in an unfavorable setting. The search game has a specific structure, where the strategy

space of the leader is large (joint actions of the patrolling units), while the strategy space of the follower is significantly smaller. This structure is favorable for BC15, since it implies a relatively small number of binary variables (the MILP contains one binary variables for each sequence of the follower), with the overall size of the MILP being linear in the size of the game, while the size of our underlying LP is quadratic due to the correlation plan.

Conclusion

We present a novel domain-independent algorithm for computing Strong Stackelberg Equilibria (SSE) in extensive-form games that uses the correlated variant of Stackelberg Equilibria (Stackelberg Extensive-Form Correlated Equilibrium). This work opens several areas for future research. First, our algorithm can be adapted and applied for solving specific domains since its scalability is significantly better in comparison to the existing algorithms. Second, the scalability can most-likely be further improved by employing iterative approaches for solving the underlying linear program. Third, several question were not addressed in our approach and remain open: Is it possible to generalize presented algorithm for computing SSE with multiple followers? Can we relax the assumption of perfect recall?

Acknowledgments

This research was supported by the Czech Science Foundation (grant no. 15-23235S), by 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 and Office of Naval Research Global (grant no. N62909-13-1-N256). This material is based upon work supported by the National Science Foundation (grant no. IIS-1253950). This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS15/205/OHK3/3T/13 and SGS15/206/OHK3/3T/13.

References

Bosansky, B., and Cermak, J. 2015. Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games. In *AAAI Conference on Artificial Intelligence*, 805–811.

Bosansky, B.; Branzei, S.; Hansen, K. A.; Miltersen, P. B.; and Sorensen, T. B. 2015. Computation of Stackelberg Equilibria of Finite Sequential Games. In *11th Conference on Web and Informatics (WINE)*.

Conitzer, V., and Korzhyk, D. 2011. Commitment to Correlated Strategies. In *AAAI Conference on Artificial Intelligence*.

Durkota, K.; Lisy, V.; Bosansky, B.; and Kiekintveld, C. 2015. Approximate solutions for attack graph games with imperfect information. In *Decision and Game Theory for Security*, 228–249. Springer.

Fang, F.; Stone, P.; and Tambe, M. 2015. When Security Games Go Green: Designing Defender Strategies to Prevent Poaching and Illegal Fishing. In *Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, 2589–2595.

Jiang, A. X.; Yin, Z.; Zhang, C.; Tambe, M.; and Kraus, S. 2013. Game-theoretic Randomization for Security Patrolling with Dynamic Execution Uncertainty. In *12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 207–214.

Koller, D.; Megiddo, N.; and von Stengel, B. 1996. Efficient Computation of Equilibria for Extensive two-person Games. *Games and Economic Behavior* 247–259.

Leitmann, G. 1978. On generalized Stackelberg strategies. *Journal of Optimization Theory and Applications* 26(4):637–643.

Letchford, J., and Conitzer, V. 2010. Computing Optimal Strategies to Commit to in Extensive-Form Games. In *11th ACM conference on Electronic commerce*, 83–92.

Letchford, J.; MacDermed, L.; Conitzer, V.; Parr, R.; and Isbell, C. L. 2012. Computing Optimal Strategies to Commit to in Stochastic Games. In *AAAI Conference on Artificial Intelligence*.

Pita, J.; Jain, M.; Marecki, J.; Ordóñez, F.; Portway, C.; Tambe, M.; Western, C.; Paruchuri, P.; and Kraus, S. 2008. Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport. In *7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 125–132.

Rabinovich, Z.; Jiang, A. X.; Jain, M.; and Xu, H. 2015. Information Disclosure as a Means to Security. In *14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 645–653.

Shieh, E.; An, B.; Yang, R.; Tambe, M.; Baldwin, C.; DiRenzo, J.; Maule, B.; and Meyer, G. 2012. Protect: A deployed game theoretic system to protect the ports of the united states. In *11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 13–20.

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

Tsai, J.; Kiekintveld, C.; Ordóñez, F.; Tamble, M.; and Rathi, S. 2009. IRIS - A Tool for Strategic Security Allocation in Transportation Networks Categories and Subject Descriptors. In *8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 37–44.

von Stengel, B., and Forges, F. 2008. Extensive-form Correlated Equilibrium: Definition and Computational Complexity. *Mathematics of Operations Research* 33(4):1002–1022.

von Stengel, B., and Zamir, S. 2004. Leadership with Commitment to Mixed Strategies. Technical report, CDAM Research Report LSE-CDAM-2004-01.

Xu, H.; Rabinovich, Z.; Dughmi, S.; and Tambe, M. 2015. Exploring Information Asymmetry in Two-Stage Security Games. In *AAAI Conference on Artificial Intelligence*.

References

[1] Tansu Alpcan and Tamer Basar. An intrusion detection game with limited observations. In *12th Int. Symp. on Dynamic Games and Applications, Sophia Antipolis, France*, 2006.

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

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

[4] Branislav Bosansky and Jiri Cermak. Sequence-form algorithm for computing stackelberg equilibria in extensive-form games. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[5] Ahto Buldas and Roman Stepanenko. Upper bounds for adversaries utility in attack trees. In *GameSec*, pages 98–117. 2012.

[6] Jin-Yi Cai, Vinod Yegneswaran, Chris Alfeld, and Paul Barford. An attacker-defender game for honeynets. In *Computing and Combinatorics*, pages 7–16. Springer, 2009.

[7] Thomas E Carroll and Daniel Grosu. A game theoretic investigation of deception in network security. *Security and Communication Networks*, 4(10):1162–1172, 2011.

[8] Pavel Čeleda, Jakub Čegan, Jan Vykopal, and Daniel Tovarňák. Kypo–a platform for cyber defence exercises. 2015.

[9] Karel Durkota and Viliam Lisý. Computing optimal policies for attack graphs with action failures and costs. In *STAIRS*, pages 101–110, 2014.

[10] Karel Durkota, Viliam Lisý, Branislav Bošanský, and Christopher Kiekintveld. Approximate solutions for attack graph games with imperfect information. In *Decision and Game Theory for Security*, pages 228–249. Springer, 2015.

[11] Karel Durkota, Viliam Lisý, Branislav Bošanský, and Christopher Kiekintveld. Optimal network security hardening using attack graph games. In *Proceedings of IJCAI*, pages 7–14, 2015.

[12] Mohammad GhasemiGol, Abbas Ghaemi-Bafghi, and Hassan Takabi. A comprehensive approach for network attack forecasting. *Computers & Security*, 2015.

[13] Russell Greiner, Ryan Hayward, Magdalena Jankowska, and Michael Molloy. Finding optimal satisficing strategies for and-or trees. *Artificial Intelligence*, pages 19–58, 2006.

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

[15] 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.

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

[17] Manish Jain, Jason Tsai, James Pita, Christopher Kiekintveld, Shyamsunder Rathi, Milind Tambe, and Fernando Ordóñez. Software assistants for randomized patrol planning for the lax airport police and the federal air marshal service. *Interfaces*, 40(4):267–290, 2010.

[18] Joshua Letchford and Yevgeniy Vorobeychik. Optimal interdiction of attack plans. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 199–206. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[19] Jingqiang Lin, Peng Liu, and Jiwu Jing. Using signaling games to model the multi-step attack-defense scenarios on confidentiality. In *Decision and Game Theory for Security*, pages 118–137. Springer, 2012.

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

[21] 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.

[22] 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.

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

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

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

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

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

[28] Mahmoud T Qassrawi and Zhang Hongli. Deception methodology in virtual honeypots. In *Networks Security Wireless Communications and Trusted Computing (NSWCTC), 2010 Second International Conference on*, volume 2, pages 462–467. IEEE, 2010.

[29] Sankardas Roy, Charles Ellis, Sajjan Shiva, Dipankar Dasgupta, Vivek Shandilya, and Qishi Wu. A survey of game theory as applied to network security. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, pages 1–10. IEEE, 2010.

[30] Carlos Sarraute, Gerardo Richarte, and Jorge Lucángeli Obes. An algorithm to find optimal attack paths in nondeterministic scenarios. In *Proc. of the 4th ACM workshop on Security and artificial intelligence*, AISec '11, pages 71–80, New York, NY, USA, 2011. ACM.

[31] Reginald E. Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. In Sushil Jajodia and Javier Lopez, editors, *Computer Security - ESORICS 2008*, volume 5283 of *Lecture Notes in Computer Science*, pages 18–34. Springer Berlin Heidelberg, 2008.

[32] Stephan Schmidt, Tansu Alpcan, Şahin Albayrak, Tamer Başar, and Achim Mueller. A malware detector placement game for intrusion detection. In *Critical Information Infrastructures Security*, pages 311–326. Springer, 2008.

[33] 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.

[34] Eric Shieh, Bo An, Rong Yang, Milind Tambe, Craig Baldwin, Joseph DiRenzo, Ben Maule, and Garrett Meyer. Protect: A deployed game theoretic system to protect the ports of the united states. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 13–20. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[35] Lance Spitzner. Honeypots: Catching the insider threat. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 170–179. IEEE, 2003.

[36] Lance Spitzner. *Honeypots: tracking hackers*. Addison-Wesley Reading, 2003.

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

[38] Ondřej Vaněk, Zhengyu Yin, Manish Jain, Branislav Bošanský, Milind Tambe, and Michal Pěchouček. Game-theoretic resource allocation for malicious packet detection in computer networks. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 905–912. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[39] Jiří Čermák, Branislav Bošanský, Karel Durkota, Viliam Lisý, and Christopher Kiekintveld. Using correlated strategies for computing stackelberg equilibria in extensive-form games. In *Thirtieth Conference on Artificial Intelligence*, 2016.