Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

Algorithm with Quality-Runtime Tradeoff Parameter for Attack Policy in Attack Graphs for Network Security

May, 2016 **Author: Elnaz Babayeva**

Supervisor: Karel Durkota

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Elnaz B a b a y e v a

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Algorithm with Quality-Runtime Tradeoff Parameter for Attack Policy Algorithm in Attack Graphs for Network Security

Guidelines:

Computing the administrator's best action to increase the network security requires numerous computations an attacker's best attack policy (reply) for each of administrator's decisions (actions). Computing the optimal attack policy in the attack graph is known to be an NP-hard problem. However, fast computation of the lower bounds (to the optimal attack policy), and iterativly improving it, can be used to direct the exploration of the administrators action and results in a faster overall algorithm. Propose and implements algorithm with parameter for tradeoff between quality and runtime of attack policy computation as follows:

1. Study the compact representation of the attacker attack space using Attack Graphs.
2. Learn the algorithm (in references) for computation the optimal attack policy.
3. Propose possible parameter(s) to control the tradeoff between quality and runtime of the attack policy.
4. Implement and compare the algorithm to current state-of-the-art algorithm.

Bibliography/Sources:

[1] Durkota, K., Lisy, V., Bosansky, B., Kiekintveld, C.: Optimal network security hardening using attack graph games. In: Proceedings of IJCAI, pp. 7-14 (2015)
[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] Approximate Solutions for Attack Graph Games with Imperfect Information. Karel Durkota, Viliam Lisý, Branislav Bošanský, Christopher Kiekintveld. GameSec 2015.

Bachelor Project Supervisor: Ing. Karel Durkota

Valid until: the end of the summer semester of academic year 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, December 11, 2015

Author statement for undergraduate thesis:

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date ________________ ______________________

signature

Acknowledgement

I would like to thank my supervisor Ing. Karel Durkota for regular and very useful consultations, helpful advice and for the time Ing. Karel Durkota has devoted to control and improve my work. I would like to thank my family and close friends for their help and support during my studies.

Abstrakt

Hledání optimální strategii v útočném grafu je považována za NP těžkou úlohu. Correlated Stackelberg Equilibrium (CSE) aproximuje Strong Stackelberg Equilibrium (SSE), který hledá optimální strategie pro dva hráče, obránce a útočníka. Tato práce se zabývá použitím iterativních algoritmů s účelem zmenšení času běhu CSE algoritmu. Iterativní algoritmy umožňují postupně zlepšovat dolní odhad optimální strategie. Tenhle způsob slouží k směrování prohledávání útočného grafu a snížení složitostí prostoru řešení. Navržená technika snaží se najít kompromis mezi kvalitou řešení a časem běhu algoritmu.

Klíčová slova

(síťové zabezpečení, teorie her, útočný graf, honeypot alokace, iterativni algoritmus, kompromisní parameter)

Abstract

Finding the optimal policy in the attack graph is known to be an NP-hard problem. Correlated Stackelberg Equilibrium (CSE) approximates Strong Stackelberg Equilibrium (SSE), which finds the optimal policy profile for two players, a defender and an attacker. This thesis investigates the application of iterative algorithms to reduce the computation time of CSE algorithm. Iterative algorithms allow improving lower bound to the optimal attack policy. It can be used to direct exploration of the attack graph and by that to reduce the complexity of the searching space. Proposed techniques look for a tradeoff parameter between algorithm's quality and its runtime.

Keywords

(network security, game theory, attack policy, honeypot allocation, iterative algorithms, tradeoff parameter)

Contents

1 Introduction **1**

2 Background **3**

2.1 Network 3
2.2 Attack Graph 4
2.3 Attack Policy 6
2.4 Markov Decision Process 7
2.5 Partial Observable Markov Decision Process 8
2.6 Closed Under Rational Behavior Set 9
2.7 Contribution 10

3 Solving Iterative Attack Policies **11**

3.1 Iterative Algorithms 11
3.2 Iterative Deepening Algorithm 12
3.3 Iterative Reward Bounding Algorithm 13

4 Experiments **15**

4.1 Iterative Deepening Algorithm 15
4.2 Iterative Reward Bound Algorithm 16
4.3 Comparison of Two Methods 18

5 Imperfect Information Honeypot Allocation Game **19**

5.0.1 Nature's Actions 19
5.0.2 Defender's Actions 20
5.0.3 Attacker's Actions 21
5.0.4 Players' Utilities 21
5.0.5 Solution Concept 22

5.1 Game Approximation 22

5.2 Problem Statement 24

6 Iterative Algorithm on HP Allocation Game **25**

6.1 Application of Iterative Algorithms into Game 25

6.2 Iterative Deepening 26

6.3 Iterative Reward Bound 26

6.4 Tradeoff Parameter 28

6.4.1 Iterative Reward Bound Completed Algorithm 28

6.4.2 Parameter Modification 29

6.4.3 Iterative Reward Bound Approach 29

6.4.4 Choice of Information Set 30

7 Experiments **31**

7.1 Network 31

7.2 Experimental Approach of Choosing Tradeoff Parameter 32

7.3 Scalability 33

7.4 Results 34

8 Conclusion **37**

Bibliography **40**

Chapter 1

Introduction

Networked computer systems have got a wide range of usage for the last decades. As a result of it, not only the term of a cyber attack was invented, but also the number of computer attacks increases every year. Recently there has been research interest in the game-theoretic approach to secure networked computer system, so we also focus on hardening networked computer system using game theory and decoy host for the attacker called honeypots.

Even for a small network there is a need to have an automated decision system to reduce the successful attacks. Attack Graph (AG) is a representation of vulnerabilities analyzing network and detecting weak places. AG represents all the known sequence of the actions, which the attacker can use to compromise the target in the networked system. With AG, we can calculate the risk of the successful attack (Noel a Jajodia, 2004).

There are game-theoretic approaches which use attack graphs to find what parts of the network are not secured and try to harden them. One of the methods to decrease the risk of the attacks is to set honeypots (HPs) into the network (Durkota et al., 2015*b*). There are two reasons to set up honeypots. Firstly, get a record of the attacker's activity, thereby to get a deep look at the methodologies of the network insecurity. Secondly, honeypots can serve as Intrusion Detection System. The main goal of a honeypot is to distract the attacker from real hosts. But it is expensive to set and assemble a believable, hardly detectable honeypot.

In paper (Durkota et al., 2015*b*), the authors use the game-theoretic model to choose what real hosts to duplicate as honeypots to develop a more secure network. The game-theoretic model proposes an interaction between two players, attacker and defender, to find the best strategies for both of them. With AG, we find an optimal plan for the attacker to get the highest reward as possible. The defender, an administrator of the

network, decides for the type of honeypot and looks for a tradeoff between the cost of a honeypot and his loss in case of a successful attack.

The current algorithm generates all relevant attack plans, which attacker can apply and then find the best host type for setting up HPs. In addition, the general case of *Strong Stackelberg Equilibria* (SSE) is NP-hard problem. Authors in (Durkota et al., 2015*a*) present linear program form of SSE approximation that finds a close solution in polynomial time. The problem of this LP formulation is that it needs all the attack plans in advance to construct LP and to find the optimal strategy for the defender. To reduce the number of attack plans, authors use only promising ones for the network.

In this work we propose an iterative algorithm for finding attack policies and iteratively build honeypot allocation game, avoiding calculating all the attack policies optimally. We introduce two iterative algorithms for AG: Iterative Deepening and Iterative Bounding Algorithm. In iterative algorithms, the defender has an optimistic belief how much he loses if the attack succeeds while the attacker has a belief that he gets less than he could. We limit the attacker's actions and find a lower bound of the attacker's best plans, then investigating what is the best choice for the defender in this iteration. Iteratively we increase the number of attacker's actions approaching to the optimal solution of the honeypot allocation game.

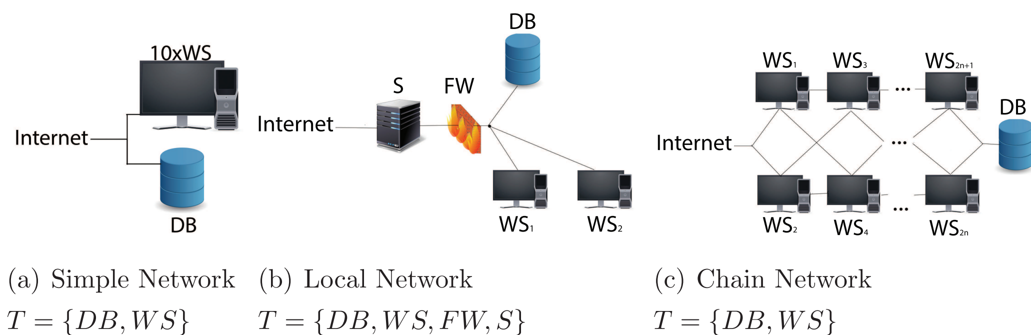
The first part of the work focuses on iterative algorithms, and it's application to attack graphs. The second part describes iterative algorithms implementation to the honeypot allocation game and choosing iterative parameter which finds a tradeoff between computational time and effectiveness of the algorithm.

Chapter 2

Background

2.1 Network

We describe a computer network as a list of host different types T, such as firewall, server, workstation, etc. Two hosts are of the same type if they run the equal services, have the same connectivity in the network and have the equivalent value for the player. All hosts of the same type produce the same attack space, presenting the same vulnerabilities of the network, so they can be represented only once in the attack graph due to scalability. During the attack a specific host of given type is chosen randomly. Formally, a computer network $x \in N_0^T$ contains x_t hosts of type $t \in T$. For example in Figure 2.1(a), the set of types is $T = (WS, DB)$ and $n = (10, 1)$, meaning that in the network there are 10 workstations (WS) and one database (DB).



(a) Simple Network $T = \{DB, WS\}$ (b) Local Network $T = \{DB, WS, FW, S\}$ (c) Chain Network $T = \{DB, WS\}$

Figure 2.1: Different simple networks.

2.2 Attack Graph

There are various representations of attack graphs. We use dependency AND-OR attack graphs because they are compact and allow better analysis. They are directed graphs, which have AND and OR nodes, where a fact is OR node and an action is AND node. Fact f is a logical statement about the network, while an action could make the facts true. Fact nodes bring a reward for the attacker. An action a has preconditions ($pre(a)$) and effects ($eff(a)$). Preconditions – a set of facts, which must be true before a can be performed, and effects – a set of facts which will be true if action a succeeds. The edges of the graph represent the relations between action and its preconditions or effects. We apply a monotonicity assumption (Qian et al., 2010), that once some fact is true, it will remain unaltered during entire attack. Every action is assigned a probability that this action succeeds and a cost — the price the attacker pays to perform the action whether the action succeeds or not. Besides, there is a set of host types the attacker interacts with, if some fact is true. Formally, attack graph can be represented like a tuple $AG = \langle F, A, T, r, p, c, hp \rangle$, where

- T is set of host types in the network
- F is a finite set of facts
    - $T_f \subset T$ is set of host types the attacker interacted with if fact f is true
- A is finite set of actions
    - Fact $f \in F$ depends on action $a \in A$ if $f \in eff(a)$ and action a depends on f if $f \in pre(a)$
- $r : F \to \mathbb{R}$ is a reward (denoted as r_f)
- hp: $A \to [0, 1]$ is a probability that action interacts with honeypot, denoted as hp_a. $\bar{hp_a} = 1 - hp_a$ is probability that action does not interact with honeypot
- p: $A \to [0, 1]$ is a probability that action succeeds, denoted as p_a. We define $\bar{p_a} = 1 - p_a$ probability that action fails
    - If $hp_a > 0$ then $p_a + \bar{p_a} = \bar{hp_a}$
- c: $A \to \mathbb{R}$ is cost of the action, denoted as c_a

For example, in Figure 2.2 there is an example of attack graph. Diamonds and rectangles are facts, where rectangles are facts which initially true and diamonds are facts initially false. Diamonds are denoted by label and set of type hosts $T_f \subset T$. Actions are rounded rectangles denoted by tuple $\langle p_a, c_a \rangle$. The attack proceeds from top to bottom. In this particular AG there is only one reward, when attacker achieves root privileges on database server (*1: execCode*). At the beginning attacker can perform following actions: *9:RULE 20* or *15:RULE 1*. If the attacker decides to infect a website (action *9:RULE 20*) he immediately pays $c_a = 4$ and with probability $p_a = 0.8$ the user will browse malicious website and fact *8:accessMaliciousInput* becomes true.

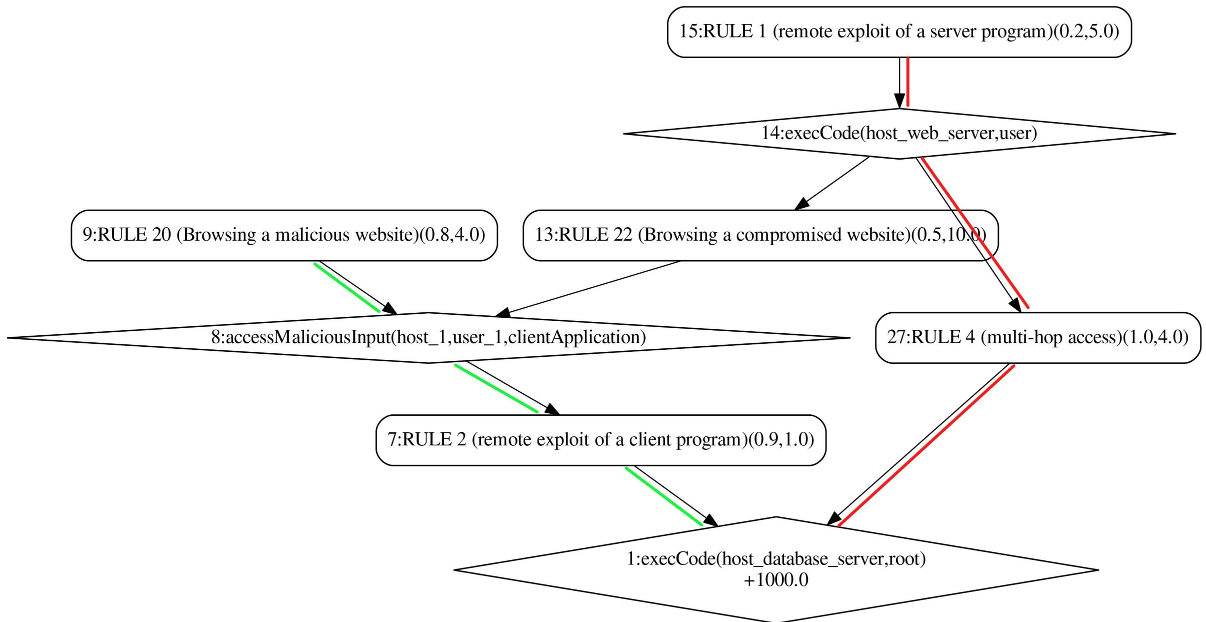


Figure 2.2: Attack Graph

There are several tools to generate the attack graphs automatically. We use $MulVAL$ [1], which builds graphs from information collected by scanning tools like $Nessus$ [2] or $OpenVAS$[3]. Costs and probabilities could be estimated using Common Vulnerability Scoring System (CVSS) in the National Vulnerability Database or directly from the administrator of some network.

[1]http://people.cis.ksu.edu/ xou/mulval/
[2]http://www.tenable.com/products/nessus-vulnerability-scanner
[3]http://www.openvas.org/

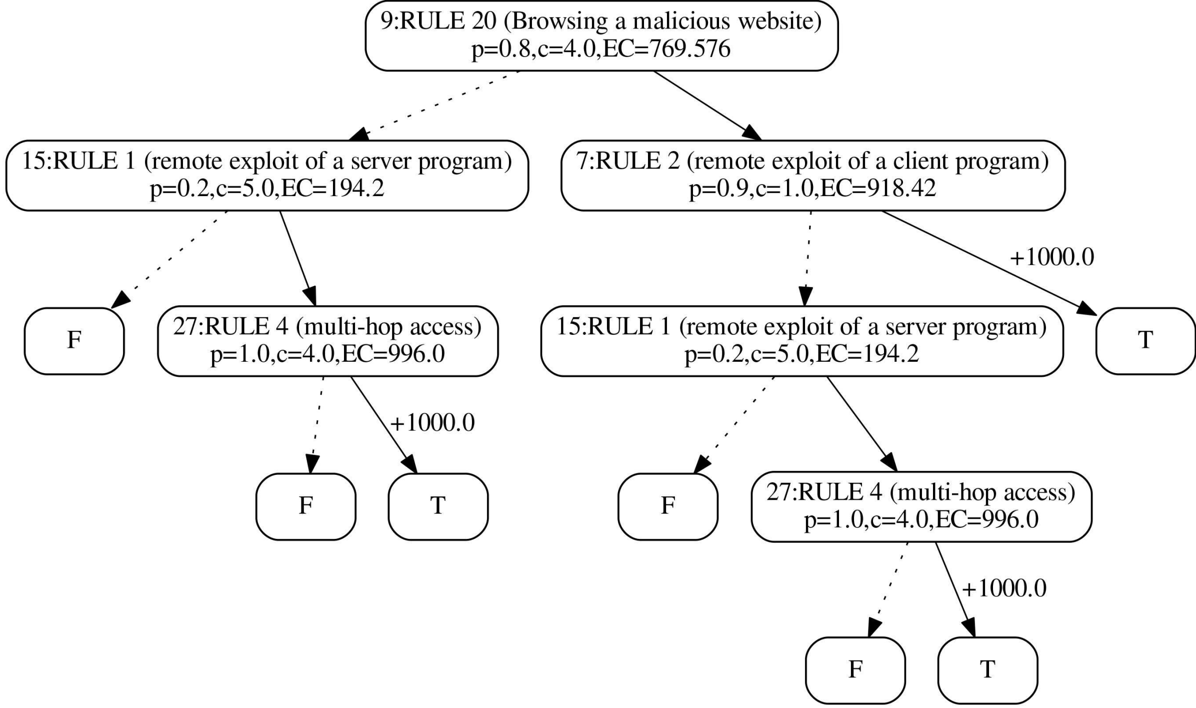


Figure 2.3: Optimal attack policy for the attack graph depicted on Figure 2.2. Solid arcs show the subpolicy, if the action is successfully executed, dashed if action fails. Also, there is expected utility EC for every action.

2.3 Attack Policy

By solving AG we mean to find attack policy, which characterizes every action during an attack. Formally, attack policy (AP) is an oriented tree ξ for evaluating AG. Nodes of attack policy are actions A. Performing action $a \in A$ has three possible outcomes, represented as edges: the action succeeds, fails or interacts with the honeypot. If the action has interacted with HP, then the attack ends. Otherwise, the attacker can execute action $b \in A$, if $\forall pre(b) \in F$ are true. We define a history of attack ζ as a list of tuples $\langle a, label\rangle$, where $a \in A$ and $label = \{true, false\}$ denoting if the action succeeds or not. History ζ constructs order of the actions being executed during the attack.

We define an expected cost of every attack policy ξ. Let χ be a node of AP and ψ_χ be a subtree of node χ. Suppose action a has been exploited at the node χ, which means that $\forall f \in pre(a)$ are true. Then ψ_{χ^+} is a subtree of success, ψ_{χ^-} is a subtree of fail, then expected utility ϑ of ψ_χ can be calculated recursively as follows:

$$\vartheta(\psi_\chi) = -c_a + (1 - hp_a) \cdot (p_a \cdot (\vartheta(\psi_{\chi^+}) + r_{eff(a)}) + \bar{p_a} \cdot \vartheta(\psi_{\chi^-})) \tag{2.1}$$

The optimal policy is a policy which brings the highest expected utility. To find the optimal policy we need to find all the policies and their expected utilities. In Formula 2.1 we need to take into account two points. Firstly, a reward could be taken only once. Assuming that there are two actions $a, b \in A$ and $f \in eff(a),\ f \in eff(b)$, $r_f > 0$. If the attacker performs action a and it succeeds, he gets a reward r_f. During an attack, he could perform successfully action b, but he does not get a reward r_f because it has been obtained by action a. Secondly, we assume that the attacker is a rational player, that he attacks according to the optimal attack policy to get the highest reward. For the first time, the attacker attacks a specific host h of type $t \in T$, which is chosen uniformly. If the attacker interacts with host type t in the future during the attack, he will interact with the same host h, because the reward is the same for every host type and probability of interacting with honeypot drops to zero. The attacker can terminate an attack at any time.

Figure 2.3 depicts the optimal AP for the attack graph in Figure 2.2 without honeypots. Nodes represent suggested actions; solid arc represent subpolicy if the action succeeds, and dot arcs represent subpolicy if the action fails. In Figure 2.3 the first suggested action is *9: RULE 20*, if it fails, the attacker's next best choice *15:RULE 1* otherwise he proceeds with *7:RULE 2*. In every action's subpolicy we calculate expected utility EC if the policy is followed. The rational attacker terminates the attack if there are no actions to perform, expected utility is lower than the cost of actions or all the rewards have been collected.

2.4 Markov Decision Process

To compute the optimal attack policy, we use a finite horizon $Markov\ Decision\ Process$, denoted as MDP. MDP represents all possible attack policies in the attack graph for a single network. In MDP, we define a set of states S. Every state $s \in S$ consists of i) set of actions A_s, which can be exploited in this state ii) $T_s \subset T$ the set of host types the attacker has interacted with and ii) $F_s \subset F$ the set of achieved facts. Every action has a probability of success, failure or interacting with honeypots. If the attacker interacts with honeypot, then the attack ends, otherwise if there are more actions to commit, a new state is generated as outcome of that action. If action a made a fact $f \in eff(a)$ true and $r(f) > 0$ then the action brings a reward. The rewards are summed and represented in

terminal states of the game. Moreover, the MDP is a tree, where nodes are states S, and edges represent actions A', probabilities of actions modeled like a chance nodes. Attacker chooses an action in each of MDP states. Formally, MDP for AG can be represented as tuple $< A', S, P, R >$, where

- $A' = A \cup \Gamma$ is set of actions, where Γ is termination action, which has $c_\Gamma = 0,\ p_\Gamma = 1,\ T_\Gamma = \emptyset$
- S is a set of states, where $s = (A_s, T_s, F_s)$
- $P^a_{ss'} \in [0,1]$ the probability that from state s with action a can be reached state s'
- $R^a_{ss'}$ is the attacker's reward for performing action a in state s heading to s', it depends on cost of action c_a, the set of facts which are true F_a and probabilities of success and interacting with honeypots.

2.5 Partial Observable Markov Decision Process

In the previous section, we describe MDP for the game tree for one network, and MDP is perfectly observed, but the defender hardens the network by setting up a honeypot. We assume that the attacker knows how many honeypots are allocated, but his ambiguity is about the type of honeypot. Therefore the attacker does not know precise structure of the network, he only has a prior belief (based on statistics) what network he observes. So the problem is translated to *Partial Observable Markov Decision Process* (POMDP). For example in Figure 2.4(a) the attacker has belief that with probability 0.4 he observes a network z_1 and with 0.6 observes network z_2. The same action in different networks can lead to the different probabilities, so for computing the probabilities in various MDP we use Bayes rule. If we are in a state s and $\beta_j(s)$ is a probability that we are in j's MDP in state s. After performing action a we occur in state s' with probability $p_j(s, s', a)$. Now, updated probability in state s' is $\beta_j(s') = \frac{p_j(s,s',a)\beta_j(s)}{\sum_{i=1}^{M} p_i(s,s',a)p\beta_i(s)}$, where M is the number of all MDPs.

(a) Network z_1

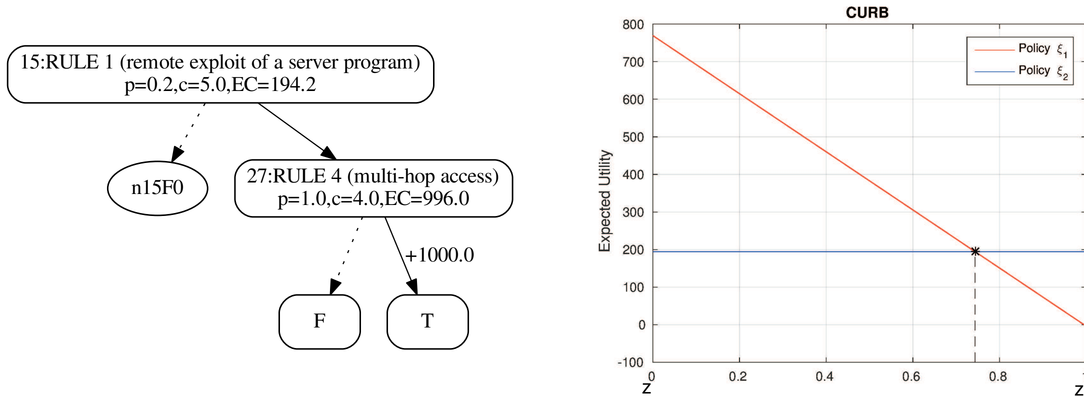
(b) Network z_2

Figure 2.4: Two networks of AG depicted Figure 2.2 with different allocation of HP. HP are denoted by yellow color. For attacker these networks are indistinguishable.

2.6 Closed Under Rational Behavior Set

For game theory analysis we are interested also in attacker's behavior in all possible beliefs among indistinguishable networks. The defender chooses how to allocate the honeypot, thereby changing attacker's belief. However in case we do not know the probability of the networks, we need to compute a set of attack policies that are optimal for some beliefs. By that, we divide attacker's belief space, and for each belief, we have his optimal attack policy.

In one attack graph, there is an exponential number of MDP attack policies. To reduce that number, we use only rationazable attack policies.A attack policy is raionazable "if and only if it is the attacker's *best response* to some belief about the network" (Durkota et al., 2015*a*). *Closed Under Rational Behavior* (later on CURB) is a set of all rationazable attack policies. For example in Figure 2.5(a) is depicted the optimal attack policy ξ_2, when attacker has a prior belief $hp = 1.0$ that HP is $host_1$ for the attack graph in Figure 2.4(a). Therefore, attacker's decision about AP depends on his belief if the honeypot is assembled in the network or not. In Figure 2.5(a) attacker has a prior belief that he observers network z, where no honeypots are placed and in z_1 where HP is $host_1$ -– network depicted Figure 2.4(a). The attacker chooses optimal strategy ξ_1, depicted in Figure 2.3, if his belief that honeypot is set less than 0.74, otherwise he chooses policy ξ_2.



(a) Optimal AP, when HP is set with $hp = 1$ (b) CURB of Attack Policies for two networks

Figure 2.5: Example of CURB

2.7 Contribution

Even having only rationazable attack policies in CURB, the scalability increases exponentially. We would like to introduce iterative algorithms which limit actions in the attacker's policies. Algorithms iteratively make CURB more complicated and iteratively it reaches to the optimal solution. This approach is needed for the honeypot allocation game, which will be described later.

Chapter 3

Solving Iterative Attack Policies

3.1 Iterative Algorithms

As it is said in "A Dictionary of Computing" by John Daintith the iterative improvement is "a technique that approaches a solution by progressive approximation" (Daintith a Wright, 2008). So iterative algorithm with every iteration approaches closely to the optimal solution. In this chapter, we propose iterative algorithms which find a lower bound (LB) of the attacker's CURB by limiting the number of the attacker's actions. It is lower bound, because with every iteration we increase number of actions, so attacker can get more chances to increase the reward and have a successful attack. Let ξ_m and ξ_{m+1} be the optimal attack policy in iterative step m^{th} and $m+1^{\text{th}}$ respectively.

Proposition 1. *Expected utility of optimal attack policy in $m+1^{th}$ iteration is at least as good as expected utility of optimal attack policy in m^{th} iteration: $\vartheta(\xi_m) \leq \vartheta(\xi_{m+1})$.*

Proof. In every iteration we increase number of actions which the attacker can exploit. In $m+1^{\text{th}}$ iteration could be action which brings a reward r_f, not obtained in m^{th} iteration. Therefore, expected utility increases. If there are not such a reward or to obtain it costs more than it's own value, then the optimal attack policy will remain the same as in m^{th} iteration.

We concentrate at the attacker's CURB and push it the optimal solution. We introduce two algorithms, which limit the attacker's actions: Iterative Deepening algorithm and Iterative Reward Bounding algorithm. Iterative Deepening algorithm limits the depth of the game tree and does not allow to exceed limited number of actions in a sequence. In every iteration, we increase the number of actions in the sequence. Iterative

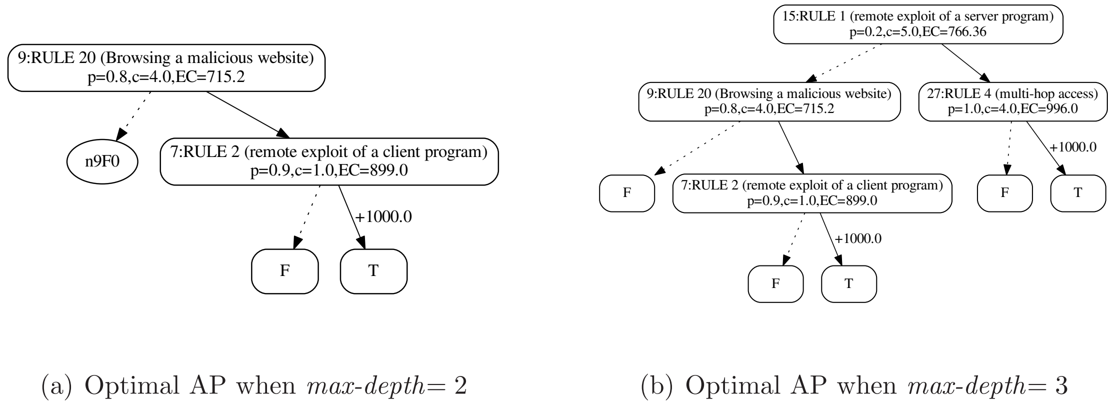
Reward Bounding algorithm prunes the states if their estimated reward is less than some chosen value. We use heuristics to estimate the reward of the state. According to the heuristics, the algorithm decides to which state the attacker should proceed further. In both of these methods there is parameter of quality, iterative step. We try to find such a parameter which will decrease runtime and number of expanded states. In Iterative Deepening it is *max-depth* and in Iterative Reward Bound it is *min-utility*. We concentrate on these two algorithms, because they have clearly definite iterative quality and with final approximation reach to the optimal solution. For example, we consider an algorithm which combines some actions into one, but then it was uncertain how to disjoin them and what price and cost should be for this action.

3.2 Iterative Deepening Algorithm

In Iterative Deepening Algorithm (later ID) we propose to limit the depth of the attack policy. The idea is that there is no policy generated with the sequence more than number of limited actions. We define *max-depth* as a variable which limits the depth of the tree. Increasing *max-depth* we increase competent sequence of actions which attacker can follow. Therefore with incrementing *max-depth*, we have more chances to get higher reward and greater expected utility of the optimal attack policy. Algorithm terminated when no states were pruned, and therefore it reached the optimal solution.

The problem is to set up iterative step, meaning how to change *max-depth* for every iteration. Attack graph is surely solved optimally when *max-depth*$= |A|$, because permitted sequence of actions will be maximum, and the attacker chooses the optimal AP among all the possible generated policies. If *max-depth* is set up to 1, all the polices will be consisted from one action, but there is no guarantee that the attacker gets a reward and his optimal strategy is not to attack. We propose an algorithm which calculates minimum number of actions in the attack policy for the attacker to get at least one reward.

In our attack graph, depicted in Figure 2.2, there are five actions which could be executed and it is the maximum sequence of actions which could be in the attack policy. In a case of this AG, there must be minimum two actions to bring a reward for the attacker. Attack policies ξ_1 and ξ_2 consisted of 2 actions are depicted in Figure 2.2 as red and green lines respectively. Calculated the expected reward for these two polices



(a) Optimal AP when *max-depth*$= 2$ (b) Optimal AP when *max-depth*$= 3$

Figure 3.1: Iterative Deepening Algorithm applied to AG in Figure 2.2

are $\vartheta(\xi_1) = 188.2$, while $\vartheta(\xi_2) = 715.2$. Optimal attack policy is ξ_2 and it is depicted in Figure 3.1(a).

In the previous example *max-depth* was two. If we increment *max-depth* by one, an optimal attack policy looks like in Figure 3.1(b). With *max-depth*$= 4$ we have found optimal policy which is depicted in Figure 2.2(b).

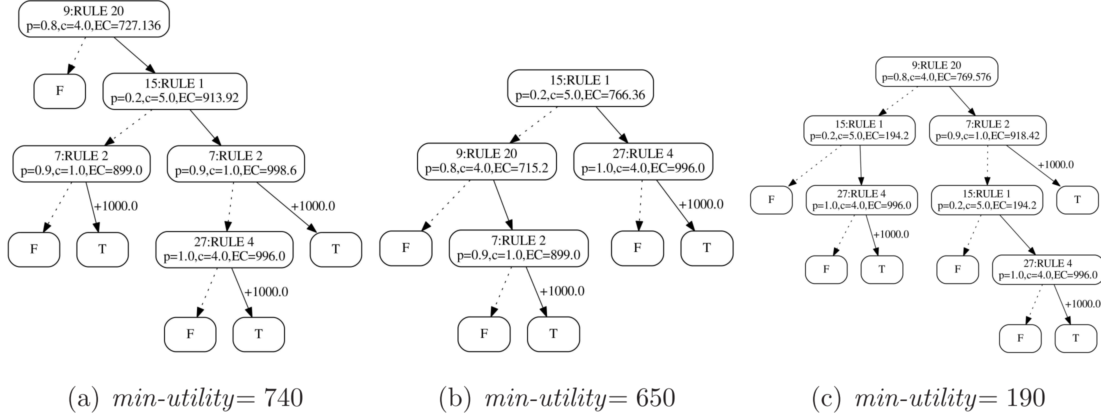
3.3 Iterative Reward Bounding Algorithm

In Iterative Reward Bounding Algorithm (IRB) we introduce an algorithm which prune the subpolicies that have a low reward for the attacker.

For state $s \in S$ we calculate heuristics $h(s)$ of expected reward. To calculate a heuristics we translate AG into attack tree, deleting edges from AG which violets tree structure. Then for every reward r_f, $r_f > 0$ we are looking for the path from r_f to the state s, calculating the cumulative probabilities of reaching the state s. Total heuristic is sum of weighed rewards. Heuristics is admissible because we do not take into account costs of the actions, so heuristics of the state does not overestimate the expected utility of the state.

Taking advantage of the heuristics, we limit the expected cost of the states by parameter *min-utility*. For every state s we calculate its heuristic $h(s)$ and if $h(s) <$*min-utility* we prune out the search and by this limit attacker's possible actions.

It is clear that the expected utility of the optimal attack policy is always higher than 0, because if it is negative, attacker chooses not to attack the network. Therefore, we find



(a) *min-utility*$= 740$ (b) *min-utility*$= 650$ (c) *min-utility*$= 190$

Figure 3.2: The optimal AP for different*min-utility for AG in Figure 2.2*

the optimal attack policy for AG with *min-utility*$= 0$, because no state are pruned. The first iterative step is with *min-utility*$= h(s_0)$, where s_0 is the first state in MDP. If all the possible actions $A_{s_0} \subseteq A$ are exploited in s_0, we get $P^a_{s_0->s'} < 1.0$, then $h(s') <$*min-utility* and AP does not expand further. So, it is the first reasonable maximum for *min-utility*, because for $w > h(s_0)$, *min-utility* $= w$, IRB cuts the s_0 right away and the optimal policy for the attacker is not to attack. The interval of *min-utility*, which can be chosen for the first state is $[0, h(s_0)]$. Algorithm terminates and finds the optimal solution when no states are pruned, so it does not always demand to be solved at *min-utility*$= 0$. In Figure 3.2 are depicted the optimal attack policies with different *min-utility* for AG in Figure 2.2.

Chapter 4

Experiments

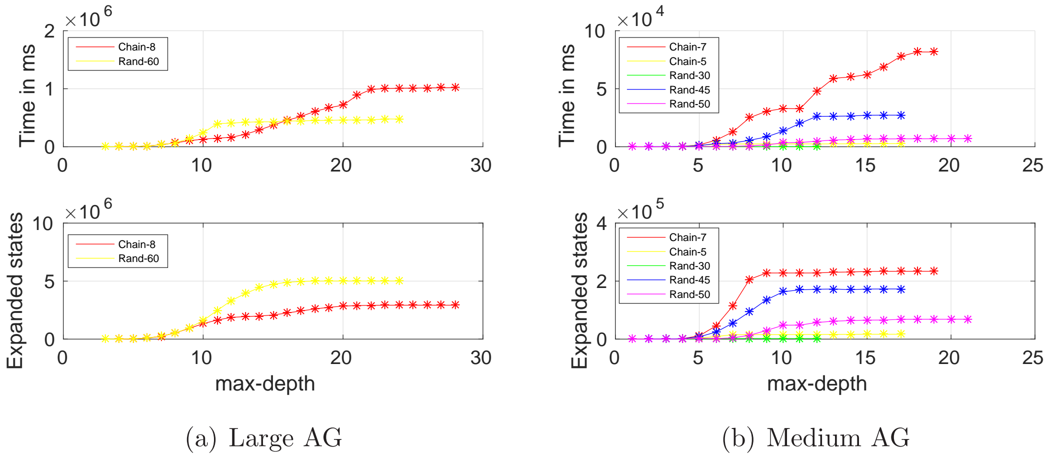
The experiments analyze the algorithms for iterative computation of the attack policies for different graphs. We would like to present the algorithm's influence on time and number of expanded states in every iteration. All experiments run on one core of Intel i5 2.4GHz processor with 3.5GB memory limit. We use two different computer structures, which are depicted in Figure 2.1(b) as *business* and in Figure 2.1(c) as *chain* network. Connections between the host types in the network correspond to actions in the attack graph. Attack graphs were generated with MulVAL, and additional cost, and probabilities were added using CVSS. They are different size and complexity, for better representation they are divided into two groups. In all attack graphs, only one host was noted as a reward, and it's price was set to 1000. Moreover, we have not hardened network with the honeypot. Graphs denoted as *Rand* were created by the random connection of the hosts.

4.1 Iterative Deepening Algorithm

In this section we evaluate ID algorithm for different networks incrementing *max-depth* by 1. It is minimal iteration step, and it shows all possible iterations and its influence on execution. As it can be seen from Figure 4.1 time and number of expanded states monotonous increasing and they are not directly proportionally. For example, after m^{th} iteration, *Chain-8* takes less amount of time than *Rand-60*, but expands about 2000000 states more. Time depends not only on the number of states but also on the complexity of the graph. Moreover, for AG *Chain-8* it takes similar amount of time to solve AG with *max-depth*$= 22$ and optimally with *max-depth*$= 28$. That is why it is important to

set iterative parameter optimally to reduce total computational time. For example, for AG *Chain-8* if iteration step increases by 1, total computational time is $t_{total} = 6722$s, while for step 3 time is $t_{total} = 4252$. Techniques how to choose optimally iterative step will be discussed in the next chapter.

The attacker's optimal strategy depends on the graph structure, action costs and success possibilities. We estimate these values for graph *Chain-7* choosing action probabilities uniformly and costs randomly from the interval from 0 to 100. The reward remains the same. As it can be seen from Figure 4.2, characteristic of the actions has a significant impact on the investigating of the optimal solution. The time interval is in range 4200 to $3.4 \cdot 10^5$ and number of expanded states is in the range from 1000 to $5.4 \cdot 10^6$. From Figure 4.2(a) can be seen, that the expected utility approaches to the optimal solution in first 5-6 iterations in most of the cases. The rest of the iterations calculates the decimal places of the expected utility or continues looking for the optimal solution by exploiting more states. For us, it is important that time and number of expanded states are monotonously increasing with some iterations.

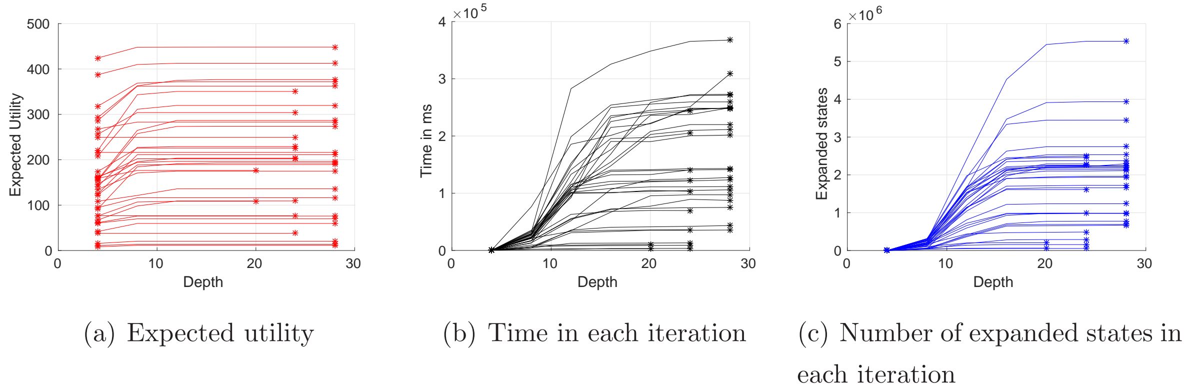


(a) Large AG (b) Medium AG

Figure 4.1: Iterative Deepening Algorithm applied to different graphs.

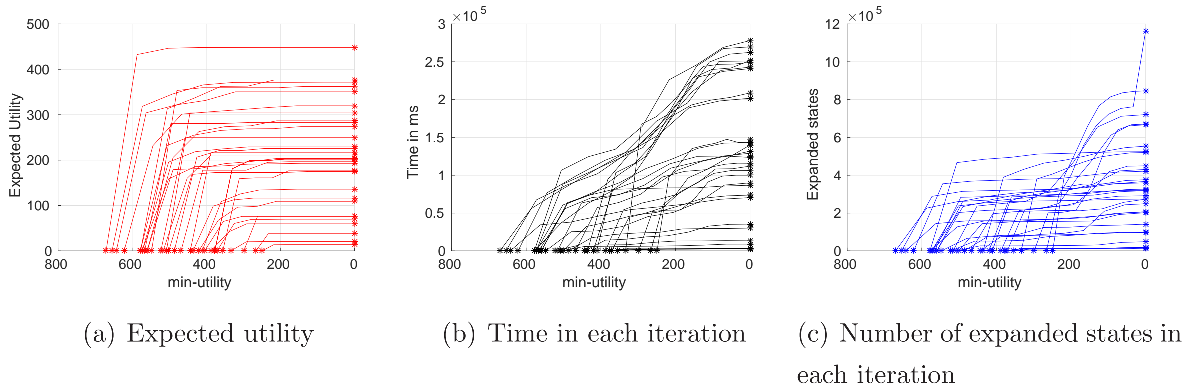
4.2 Iterative Reward Bound Algorithm

The same graphs were used for Iterative Reward Bound Algorithm. In IRB, it is not clear how to decrease *min-utility* for each iteration. Attack graph is solved optimally, when *min-utility*$= 0$. For better analysis we reversed x-axis to have a clear vision that



(a) Expected utility (b) Time in each iteration (c) Number of expanded states in each iteration

Figure 4.2: Iterative Deepening Algorithm applied to Chain-7.



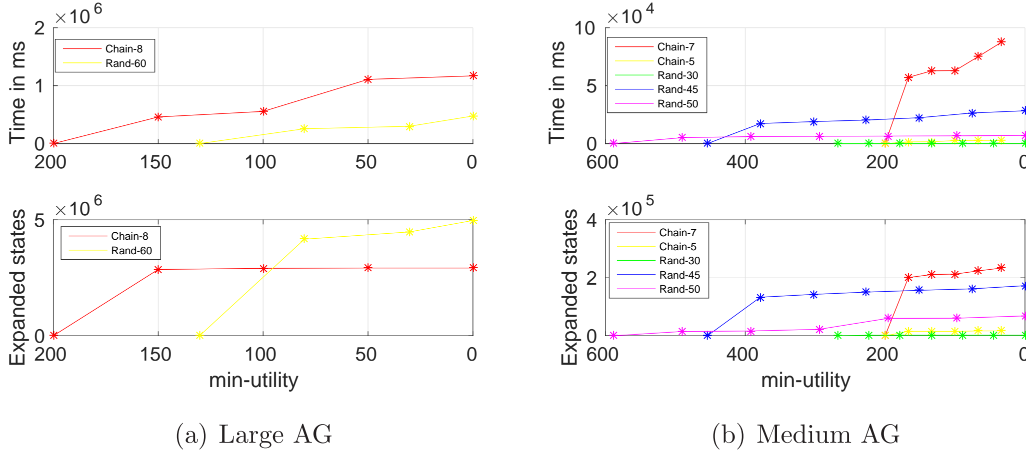
(a) Expected utility (b) Time in each iteration (c) Number of expanded states in each iteration

Figure 4.3: Iterative Reward Bound Algorithm applied to Chain-7.

time and expanded states increases as *min-utility* reaches zero. First *min-utility* is chosen depending on heuristics of initial state s_0. We have used two different iteration steps: constant and depending on heuristic. In Figure 4.4(a) are depicted graphs, where in every iteration *min-utility* was decreased by 50. For example, if in m^{th} iteration *min-utility*$= 473.5$, then in $m + 1^{\text{th}}$ *min-utility*$= 473.5 - 50 = 423.5$. And in Figure 4.4(b) are graphs, where *min-utility*$_{m+1}$ $=$*min-utility*$_m$ $- h(s_0) \cdot 0.1666$. For *Chain-8* it takes 5 iteration to solve optimally and approximately all the states were exploited in the first iteration. If in *Chain-8* we set iteration step to 50, meaning *min-utility*$_{m+1}$ $=$ *min-utility*$_m$ -50, overall time is 3174s while for *min-utility* depicted on Figure 4.4(b) it is 3494s. The problem is converted to set up optimally iteration step.

We also investigate graph *Chain-8* substituting different probabilities and costs as we have done for ID algorithm. As in ID algorithm, IRB algorithm reaches close approximation to the optimal solution in the first couple of iterations. For example, if we take the highest graph in Figure 4.3 it converges to the optimal solution on the first iteration

and then inspecting all other possible ways to get a reward.



(a) Large AG (b) Medium AG

Figure 4.4: Iterative Reward Bound Algorithm applied to different graphs.

4.3 Comparison of Two Methods

As it can be seen from the graphs depicted in Figure 4.1 and Figure 4.4, computational time and number of expanded states is not significantly different from each other. But taking into account the graph *Chain-7* ID algorithm expands in average five times more states than IRB algorithm and takes 33% more computational time. We suppose that IRB algorithm will give us better results than ID. In the next chapter, we intend to adopt these algorithms into honeypot allocation game, introduce some modification of IRB algorithm and choosing optimal iteration step.

Chapter 5

Imperfect Information Honeypot Allocation Game

Our game is an extensive two-player game with imperfect information. In this game, the attacker has prior beliefs about basic network topology, what is not always true in the real world problems. We assume that he has knowledge about the subset of host types used in the network, referring to it as a basis for the network: e.g. server, firewall, workstation. In this extensive form game, a chance player is called nature. Nature selects a network from the set of the possible networks, extensions of the basis. It is common knowledge for the defender and the attacker. The defender observers actual network and hardens it by adding honeypots. The attacker monitors the resulting network, after nature's and defender's actions, and attacks it according to the optimal policy in the attack graph of this network. After honeypot placement, different networks could look indistinguishable to the attacker. For example Figure 2.1(a) shows a possible network, and Figure 5.1 depicts a full game for this network.

5.0.1 Nature's Actions

In the network we define T as a set of host types, $n \in N$ as a total number of hosts and b basis network. Network basis describes the content of a network. X is the set of possible networks, which must include basis frame and $n - |T|$ hosts in addition. In Figure 2.1(a) the set of host types is $T = \{DB, W\}$, where DB-database, W-workstation and we set the basis is $b = (1, 0)$, which means that there is only a database in the network. Setting up $n = 2$, generated set of basis extension is $X = \{(1, 1), (2, 0)\}$. Nature choose $x \in X$, with uniform probability $p = 0.5$.

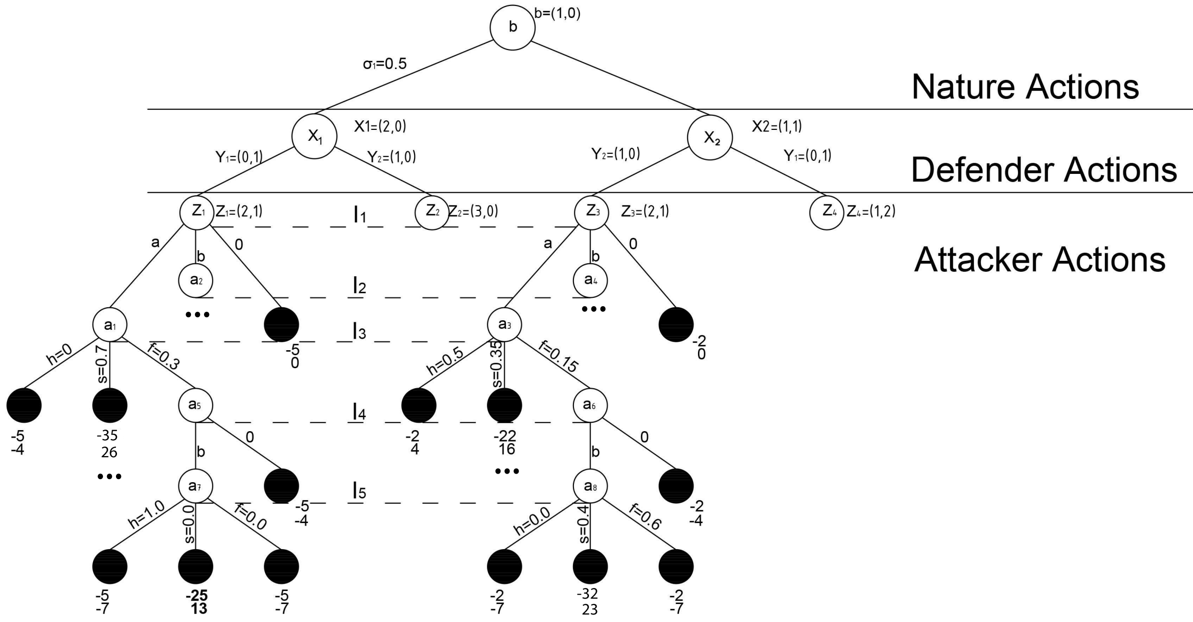


Figure 5.1: Simple game tree with $|T| = 2$ host types, basis b=(1,0), n=3 and k=1 HP. There are two actions a and b with costs $c_a = 4$ and $c_b = 3$ resp.; rewards r_a=30 and r_b=20; and success probabilities $p_a = 0.7$ and $p_b = 0.4$. Costs of assembling honeypot $c_a^h = 5$, $c_b^h = 2$.

5.0.2 Defender's Actions

The defender, player d, can put k honeypots of type T where $k \in N$ into every network $x \in X$. Set of the defender's actions is $Y = \{y \in N_0^T | \sum_{t \in T} y_t = k\}$. As a result, we receive a set $Z = X \times Y$, combination and confluence of sets X and Y. Z is a set of all networks created by nature's $\forall x \in X$ and defender's action $\forall y \in Y$, where each host of type t consists of x_t real hosts and y_t HPs. Therefore, total number of type hosts t in the network z is $z_t + y_t$. The attacker's probability of interacting with a honeypot in the host t is $\frac{y_t}{x_t + y_t}$. In addition, there is a defined cost for every honeypot $c_t^h \in R_+$, the price defender should pay to install and assemble the HP of the type t. In Figure 5.1 defender adds $k = 1$ honeypot, thereby Y={(1,0), (0,1)}. Resulting set Z consists of 4 different networks Z={(2,1), (3,0), (2,1), (1,2)}, in the first two elements the HP is in database, in others in workstation.

5.0.3 Attacker's Actions

The attacker, player a, has imperfect information about the network. Assuming that he can observe the number of hosts of each type, but he does not distinguish real hosts from honeypots. Information sets are used to model the imperfect observation. Networks in information set are identical for the attacker. If two networks $z, \dot{z} \in Z$, $z = (x, y)$ and $\dot{z} = (\dot{x}, \dot{y})$, $\forall t \in T$ $\dot{x}_t + \dot{y}_t = x_t + y_t$, then $z, \dot{z}$ belong to one information set I $\in I$. Two networks $z, \dot{z} \in I$ have the same structure of attack graphs, and differ only in the probability of interacting with HP, which has an influence on success probability and total expected utility. In other words, the attack policies generated in the network z and $\dot{z}$ are the same ($\xi_z = \xi_{\dot{z}}$). Later, we denote different networks as game states. In the Figure 5.1, the attacker observes three information sets, two of them are singletons {(3,1), (1,2)} and one information set consists of two networks $I_1 = \{[(2,0),(0,1)],[(1,1),(1,0)]\}$, where the first coordinates (2,0),(1,1) are the choice of nature and the second (0,1), (1,0) are defender's actions how to place HP. Executing the attack policy heads to the end of the game, terminal states. For example, for information set I_1 the attacker chooses action a. In our example, there is one reward, and if action a succeeds then the attacks finishes. With probability $\bar{p_a} = 0.3$ the action a fails and then the attacker executes action b, if it also fails, the attack is finished, because there no more actions for the attacker.

5.0.4 Players' Utilities

To compute the players' utility in terminal states in each leaf l we need three components: R_l-the sum of rewards for successfully executed hosts which were on the way from the root to this terminal state l; C_l the sum of the costs of every action, which was performed along the path; and H_l the defender's price of setting honeypots along the path. Moreover the defender utility is computed by $u_d(l) = -R_l - H_l$ and the attacker's utility $u_a(l) = R_l - C_l$. The utility of the attack policy is the expected utility of the terminal states. In our example, every leaf of the game tree has two values. The defender's utility is the top value, and the attacker's utility is the bottom one. For example let compute leaf l_1, marked bold in the network z_1. Action b brings a reward $r_b = 20$ and costs $c_b = 3$. Before the action b, the attacker has failed to exploit action a. The total cost $C = c_a + c_b = 7$. In the network z_1 honeypot's type is database and $c_a^h = 5$. Therefore, $u_d(l) = -20 - 5 = -25$ and $u_a(l) = 20 - 7 = 13$.

5.0.5 Solution Concept

We define Stackelberg solution concept for this game where a leader is a defender and a follower is an attacker. The defender commits to a publicly known strategy and the attacker plays the best response to the strategy of the leader. Also, we follow an assumption of breaking the ties of the leader (Yin et al., 2010). We try to maximize the defender's utility on the assumption, that the attacker will play a best response.

We determine the standard definitions from the extensive-form game. The set of actions which player follows in the game is called *strategy*. Any strategy must be complete meaning that in every uncertain state player decides what action to play. *Pure strategy* π_i assigns every information set I_i exactly one action. Π_i is a set of all pure strategies of player i. Formally, $\pi_i : h_i \rightarrow a_h$, where $h_i \in H,\ a_h \in A(h)$. *Mixed strategy* δ_i is a probability distribution on pure strategies, $\delta_i = \triangle \pi_i$, where $\pi_i \in \Pi_i$ and $\triangle$ is a probability distribution (Bosansky a Cermak, 2015). A strategy dominates another strategy if it always gives a greater utility to the player, regardless of what the other players are doing.

Best response is the strategy or strategies which produce the most favorable outcome for a player with regard to other players' strategies (Fudenberg a Tirole, 1991). Let denote set Δ_d (resp. Δ_a) as a set of mixed strategies of player d (resp. a). If the player d has a strategy $\delta_d \in \Delta_d$, then the player a, playing best response (later on τ) chooses strategy δ_a, where $\delta_a \in \Delta_a$ and $u_a(\delta_a) \geq u_a(\bar{\delta_a})\ \forall \bar{\delta_a} \in \Delta_a$ and $u_j(\delta_j)$ is the utility function, if the player follows this strategy (Osborne, 2004).

For our game Stackelberg Strategy profile is:

$$(\delta_d, \pi_a) = \operatorname*{argmax}_{\delta'_d \in \Delta_d, \pi'_a \in BR_a(\delta'_d)} u_d(\delta'_d, \pi'_a) \tag{5.1}$$

5.1 Game Approximation

The general case of computing SSE of imperfect information game is NP-hard. Also, the game tree grows exponentially with increasing number of honeypots, hosts or types of hosts. We will use the approximation of the SSE that finds a close solution to it in polynomial time. In (Bosansky a Cermak, 2015) authors present a linear program (LP) to compute SSE of a game matrix. The LP finds a probability distribution over maximum utilities for the defender, considering that the attacker plays best response. In (Durkota et al., 2015*b*) authors present formulation of extensive-form game into LP. Matrix game

is constructed for every information set. Therefore, our game is presented as a collection of matrices, and formulated as a single LP problem.

In Figure 5.2 is shown the translation from extensive-form game into two normal-form games. There are two information sets I_1 and I_2. The nature plays action x_1 (resp. x_2) with probabilities σ_1 (resp. σ_2). Therefore, the defender plays y_1 or y_3 in x_1, while y_2 and y_4 in x_2. The attacker has three various attack policies s_1, s_2, s_3, but each of them has a different probability of success. The probability of the matrix game is assigned at the leaves of the game tree. In addition the probabilities of attacker strategies should sum to the nature probabilities, meaning that p_1 through p_3 should sum to σ_1.

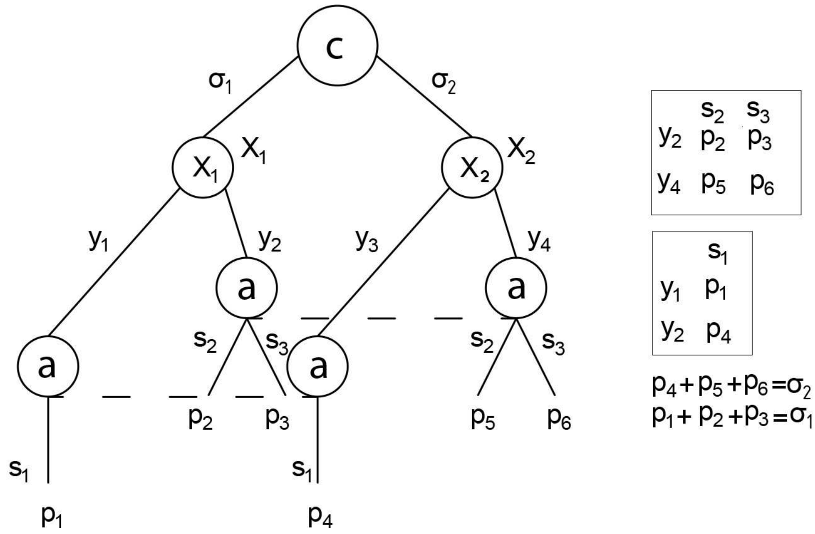


Figure 5.2: Translation of extensive game into two normal-form game

The problem of this LP formulation is that it needs all the attack policies in advance to construct LP and find the optimal strategy for the defender. In one attack graph there is exponential number of MDP attack policies, therefore we consider only rationazable strategies from CURB for every information set. In CURB there all attack polices which must be in SSE, because any attack policy in SSE is set of the attacker's best response, so it must be rationazable, and if it is, it must be in CURB. Moreover, authors in (Durkota et al., 2015*a*) present CSE algorithm, which use LP formulation and *incremental pruning alogrithm* (IPA) (Cassandra et al., 1997). IPA is an algorithm of backward induction that in every attacker's state propagates the CURB set for the part of POMDP that begins in that particular decision state. Formally, let state s has a set of actions A. CURB as a set of attack policies Ξ_a is explored $\forall a \in A$ for the part of POMDP after the action a is exploited. After committing action a in state s there is a attack policy ξ_b, which extends the set of attack policy $\xi_b \in \Xi_a$. Then from $\bigcup_{a \in A} \Xi_a$

we calculate a CURB for state s. And this algorithm is done recursively for every state. Solving the LP formulation, we find a mixed strategy for the defender: the marginal distribution over all the possible defender's actions.

5.2 Problem Statement

As it is mentioned before CSE algorithm needs to know all the attack policies in advanced. For larger games calculating all attack policies takes great amount of time and memory. In this work, we try to reduce the computation of attack policies and calculate only the relevant policies for some information sets.

The idea is after every iteration, solve LP and give the defender opportunity to decide what game states are the best strategy for him and expand further only some information sets.

We consider defender's Upper Bound CURB which we refine by iterative methods to the optimal solution. If we considered Lower Bound, CURB could never reach the optimal solution because there could be some information sets I, which has low expected utility in some iteration, while they are part of the optimal solution. Considering Upper Bound (UB), we always choose the most promising information sets with the highest expected utility of their optimal strategies. We guarantee to reach the optimal solution because u_d decreases in every iteration and LP considers maximizing overall u_d. As it is mentioned above, $u_d = -H - R$ and $u_a = R - C$, where H is constant. Because we look for defender's UB, R should get higher with an increasing number of iterations. Therefore, total attacker's utility increases, and in $m + 1^{\text{th}}$ iteration he gets higher utility than in m^{th}. We neglected the cost of the actions in the attack because the attacker plays best response. So, the problem of defender's UB can be converted into attacker's LB.

The idea is to limit the number of attacker's actions and iteratively increase them. By this, we are getting LB for the attacker, because with every iteration he has greater chances to receive a reward. In every iteration, we propose not to refine all information sets, but only in the promising ones. We would apply Iterative Deepening and Iterative Reward Bounding algorithms to reduce solving all information sets optimally.

Chapter 6

Iterative Algorithm on HP Allocation Game

6.1 Application of Iterative Algorithms into Game

The number of information sets monotonous increases with a number of honeypots, hosts or host types. As it is mentioned before, in every game state g in the information set I the attack policies are identical, the only difference is expected utilities of attack policies in various game states. In general, LP algorithm looks for marginal distributions of the most promising networks to put the honeypots in order to get a minimal loss. CSE demands the solution of each information set and expected attacker's utility of the policies in all the game states in the information set. Formally, let's say that the game has set I of information sets $I = \{I_1, I_2, I_3, I_4\}$, and game states $G = \{g_1, g_2, g_3, g_4, g_5, g_6\}$, where $\{g_1, g_3\} = I_1$, $\{g_2, g_4\} = I_2$, $\{g_5\} = I_3$, $\{g_6\} = I_4$. We run CSE algorithm, solving all the information sets optimally. It return a set of game states $G_{opt} = \{g_1, g_5, g_3\}$, where G_{opt} is the best solution for the defender to place honeypot to get minimum loss. On the one hand, we have found the optimal strategy for the defender, on the other hand it was unnecessary to solve optimally information sets I_3 and I_4, because they do not influence the solution.

Inspiring by this idea that we do not have to solve all the information sets optimally we suggest to solve every information set iteratively by ID and IRB algorithms. For example, if we limit all the attack policies by two actions and solve LP, the solution will be $G_1 = \{g_1, g_2, g_3, g_4\}$. Then we refine information set I_1 and I_2, allowing them to increase the number of actions in the attack policies. Solving LP again, we can get

another game states $G_2 = \{g_1, g_6\}$ and continue refining different iterative sets I_1, I_4 until we do not find the optimal solution.

Therefore, information sets for $m+1$ iteration are chosen from the solution of LP algorithm in m^{th} iteration. LP finds a solution in polynomial time. Therefore, it is not so time-consuming to run it after every iteration. Eventually, some information sets have to be solved optimally to make sure that we have found the optimal solution of this HP allocation game. Our intention is to set the refinement parameter *param* (*min-utility* and *max-depth* discussed in Part 1) in order to solve as less information sets optimally as possible. By this, we reduce computational time and number of expanded states.

In Algorithm 1 is shown pseudocode of the application iterative algorithms to the game. As it can be seen we do not refine all the information sets, but only these which are the best for the defender according to the attacker's optimal attack policies in the current iteration. Let denote information set *satisfiable*, if LP chose it in the previous iteration.

6.2 Iterative Deepening

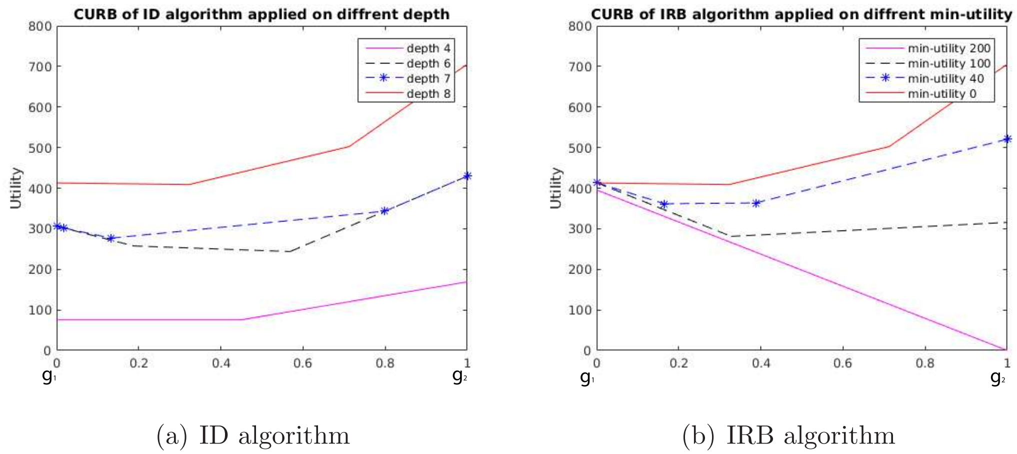
In ID algorithm, we iteratively increment the depth of attack policies in the information sets. Formally, let I be the set of all information sets of the game. For every information set we find minimum number of actions *min-depth* to get a reward (this is $param_1$ in Algorithm 1). Therefore, we get a pair $\langle I_k, param_1 \rangle$, where $I_k \in I$ and $param_1$ is the minimum depth of attack policy in I_k to get a reward. The first iteration is $\forall I_k \in I$, we run ID with corresponding parameter *max-depth*$= param_m$. Then we increase *max-depth* for *satisfiable* set. The main contribution is how to set up a parameter to decrease total computational time, which will be described later for both algorithms. In Figure 6.1(a) is shown the refinements of the CURB over different depth of one particular IS. As it is seen, increasing the depth CURB tides to optimal policies, depicted as a red line.

6.3 Iterative Reward Bound

IRB algorithm for AG prunes state, if $h(s) <$*min-utility* and then monotonously decreasing *min-utility* in order to reach optimal solution. For this game, we change heuristics

Input: $Set < InformationSet >$ I
Output: $Map < GameState, Probability >$ defenderStrategy
$defenderStrategy \leftarrow Map < GameState, Probability >$;
$strategiesSet \leftarrow Set < Strategies >$;
for *each* I_k *from* I **do**
  $param_1^{I_k} \leftarrow$ tradeoff parameter of the first iteration for I_k information set;
  $strategiesSet.add(IterativeAlgorithm(I_k, param_1^k))$;
end
$ISToRefine = LP(strategiesSet)$ returns IS which are the best choice for the defender in this iteration;
while *all* I *from* $ISToRefine$ *are not solved* **do**
  for *each* I_k *from* $ISToRefine$ **do**
    if I_k *is not solved* **then**
      $param_m^{I_k} \leftarrow$ a trade off parametr fo IS I_k in m^{th} iteration;
      $param_{m+1}^{I_k} = changeParametr(param_m^{I_k})$;
      $strategiesSet.add(IterativeAlgorithm(I_k, param_{m+1}^{I_k}))$;
    end
  end
  $ISToRefine = LP(strategiesSet)$;
end
$defenderStrategy \leftarrow$ from last $ISToRefine$ the defender optimal game state and probability;

Algorithm 1: Iterative Refinement Algorithm



(a) ID algorithm

(b) IRB algorithm

Figure 6.1: Iterative refinement of CURB set.

because now we have to take into account beliefs about the network. Now heuristic is defined by the state and the game state in IS. First parameter for information set I_k is $min\text{-}utility_{I_i} = max(h(g_j(s))$, where $g_j \in I_i,\ s = s_0$. For some $s \in S$, difference in the value of heuristics in various game states could be significant. Therefore, we have chosen maximum for all game states to avoid extra deepening to the game tree. Our algorithm returns the CURB of attack policies. If there is the set of $\{g_1, g_2..g_i\} \in I_i$, and g_i is pruned, meaning that $h_{g_i}(s) <$*min-utility*$_{I_i}$, then for $g_{-i} \in I_i$,where $h_{g-i}(s) >$*min-utility*$_{I-i}$, we continue calculating attack, while for g_i attack ends in state s and returns current policy. For satisfiable information sets, we decrease *min-utility* to get closer to the optimal solution. As it is mentioned above, IS is solved optimally when *min-utility*$= 0$. In Figure 6.1(a) is shown the CURB's refinements with different *min-utility*. Decreasing *min-utility* the CURB is approaching to the optimal solution.

6.4 Tradeoff Parameter

The main issue in iterative algorithms is that we have to solve the information sets for a couple of times and some of them completely to reach the optimal solution. In other words, it increases overall computational time. To find a compromise between reducing the total number of expanded states and computational time we suggest to use some techniques of choosing the iterative parameter, which information sets should be updated and some modernization of algorithm.

6.4.1 Iterative Reward Bound Completed Algorithm

We introduce an approach for Iterative Reward Bound algorithm to stop the attack for all the game states in IS if one of them has been pruned. Let $g_i \in I_i$ has been pruned. IRB continues to expand policy further for other states $g_{-i} \in I_i$, where $g_{-i)\neq=g_i}$. Here we suggest to terminate the attack for all $g_i \in I_i$, if g_i does not satisfy the condition $h_{g_i} <$*min-utility*. Experiments show that this modification **IRBC** has a greater success than IRB because it immediately terminates the attack and returns current optimal attack policies for every game state.

6.4.2 Parameter Modification

First of all, we can increase or decrease iterative parameter constantly, like it was shown for AG in the first part. The method is denoted as **Const**. Formally, if I_m in m^{th} iteration has the parameter set by $param_m$ and has not been solved, in $m+1^{\text{th}}$ iteration $param_{m+1} = param_m + const$, where $const > 0$ for ID algorithm and $const < 0$ for IRB. This method is insufficient because, for different AG, optimal $const$ is different.

Also, we examine another approach which changes the parameter due to the number of times we have chosen some IS - **proportional distribution method (PD)**. For this method we define a tuple $\langle I_i, n, param_{I_i(m)} \rangle$. Suppose that CSE algorithm chooses IS I_i in every iteration. We expand this information set m_{opt} times until it is solved optimally. This information set is promising because the defender has the intention to play it in every iteration. Therefore, to decrease computational time, we change iterative parameter $param_{I_i(m)}$ proportionally to m, where m is number how many times the defender has chosen I_i. Then, every m is assigned a value modification coefficient $\theta_m \in [0,1]$, initializing how should be changed the iterative parameter. For the next iteration $m+1$ for I_i, $param_{I_i(m+1)} = param_{I_i(m)} * \theta_m$. We define a threshold m_{thresh} for n. If defender plays I_i m_{thresh} times, IS is solved optimal. By that, we avoid extra deepening.

6.4.3 Iterative Reward Bound Approach

Another approach distinguishes game states in one information set for IRB and IRBC algorithms, denoted as **GS** method. Let $I_i \in I$ have set of the game states $I_i = \{g_1, g_2..g_i\}$. Every $g_j \in I_i$ has different network topology and placements of HPs. Therefore, for some $s \in S$, $h_{g_j}(s) \neq h_{g_i}(s),\ i \neq j$, because the probability of approaching rewards are various in the game states g_j and g_i. Initially, IRB has one *min-utility* for every IS. In GS we propese a modification of IRB, which has various *min-utility* for different game states. Formally there is a tuple $\langle \textit{min-utility}_g, g \rangle$, where $\forall g \in I_i,\ I_i \in I$. This reformation is valuable, because considering one *min-utility* for all game states, can lead to the situation where in g_1 IRB has already found optimal solution, while in g_2 it is still needed more iterations to reach it. It allows us to find the optimal solution for every game state more uniformly.

In Iterative Reward Bound algorithm we prune states according to heuristics. We propose a technique, denoted as **H**, that uses the heuristics value of the pruned state

in the following iteration. Consider that state s in $g_i \in I_i$ was pruned, therefore $h_{g_i}(s) < \textit{min-utility}_{g_i}$. After some iterations, I_i is again satisfiable. Then $\textit{min-utility}_{g_i} = f(h_{g_i}(s))$, where f is decreasing modification function, for example application of $Const$ or PD methods. This approach allows us not to change parameter blindly, but have real value, when the algorithm terminated in the previous iteration.

6.4.4 Choice of Information Set

We have tried to keep track of how much defender's belief is put to information sets. We suggest an approach **limiting number of iteration sets (LIS)**, which does not take all the information sets chosen by LP, but only the most promising ones. As promising, we denote those which have the greatest probability to be selected by the defender. LP algorithm returns a tuple $\langle g, p_d(g) \rangle$, where g is a game state, $p_d(g)$ is the probability to choose this game state by the defender. After every iteration we calculate the total probability $p_d(I_i) = \sum_{g_i \in I_i} p_d(g_i)$. We expand further v information sets with greatest $p_d(I_i)$ among all the IS .

Chapter 7

Experiments

We experimentally evaluate and compare our iterative algorithms and application of different techniques to them. Also, we compare runtime of iterative approaches and CSE algorithm. Firstly we examine all the techniques in one network. Secondly, we apply the best one on different graphs and make some comparison. All experiments run on one core of Intel i5 2.4GHz processor with 3.5GB memory limit.

7.1 Network

We use two different computer network topologies, which are depicted in Figure 2.1(b) as *business* and in Figure 2.1(c) as *chain* network. Connections between the host types in the network correspond to actions in the attack graph. Actions' success probabilities p_a are generated using CVSS. Action costs c_a are chosen randomly from the interval [0,100], the host type values r_t and the honeypots' costs c_h^t are chosen proportionally, more valuable host type has higher price for assembling honeypots. The basis b of the network for a business network is set $\{S,\ DB,\ WS\}$ and for a chain is $\{WS,\ DB\}$. We scale the network by adding more host types, a number of hosts and honeypots.

7.2 Experimental Approach of Choosing Tradeoff Parameter

In the previous chapter, we introduce some approaches of choosing the parameter for iterative step. We tested them on the same network with five different host types, seven hosts and one honeypot. It the first part of the experiments, we set up the parameters randomly, choose the best approach and continue investigating it further.

In Table 7.1 we introduce a solution of the best methods from the previous chapter implicated on IRBC and ID algorithm. Empirically IRB algorithm was insufficient, it consumes 45% more than CSE algorithm and prune maximum 10% of states. IRBC algorithm is the most effective algorithms among all. It prunes about a quarter of states and contributes less time than ID algorithm.

The best technique for IRBC algorithm is mixture of percentage distribution (PD) and heuristics (H) approach. In PD method a parameter represents a list of coefficients $list = [a_1, a_2..a_n]$ how to change *min-utility*, where $a_i > a_{i+1}, a_i \in [0, 1]$. These restrictions are set up in order to reach optimal solution with increasing number of iterations. Before the iterations $\textit{min-utility}_0(I_i) = max_{g_i \in I_i}(h_{g_i}(s_0))$ for all information sets. If information set I_i was chosen by the defender m times, then $\textit{min-utility}_m(I_i) = a_m \cdot \textit{min-utility}_0(I_i)$. For this network, the best choice for PD is a list $PD_1 = [0.8, 0.4, 0]$, meaning that if IS I_i has been chosen two times, in the third iteration it is solved optimally. Combination of PD and H sets up *min-utility* not by the heuristic of an initial state, but by the heuristic of the state which has been eliminated. If state s in game state g_i is pruned we calculate heuristic $h_{g_i}(s)$ for all $g_i \in I_i$. If in m^{th} iteration I_i is again satisfiable then *min-utility*$= a_m \cdot h_{g_i}(s)$, where $a_m \in list$ from PD approach. This combination implicated on IRBC algorithm shows better time than CSE and prunes 27% of the states.

Another satisfiable approach is a union of PD and GS methods. The only difference from PD method, that we distinguish $g_i \in I_i$ and respectively the heuristic of s_0 in different game states. Therefore, $\textit{min-utility}_0(g_i) = h_{g_i}(s_0)$. From PD follows that if I_i is satisfiable in m^{th} iteration then $\textit{min-utility}_m(g_i) = a_m \cdot \textit{min-utility}_0(g_i)\ \forall g_i \in I_i$. This method consumes in significantly more time than CSE and prunes 20% of states.

In LIS method after every solution of LP, we chose only v promising information sets for the further expansions. In current example for m^{th} iteration we set up $v = |I'|/2$, where I' is set of information sets which are the optimal solution for the defender in m^{th} iteration. We combine it with previous best methods: H+PD and GS+PD. Union of

Algorithm	Method	Change of parametr	Time in ms	Expanded states
IRBC	$Const_1$	150	91254	7956
	$Const_2$	min-utiltity/2	95321	7931
	PD_1	[0.8, 0.4, 0]	54763	7361
	PD_2	[0.8, 0.5, 0.25, 0]	63215	7923
	GS+PD_1		52501	6952
	H+PD_1		50087	6453
	LIS+H+PD_1	$\lvert I'\rvert/2$	138508	6961
	LIS+GS+PD_1	$\lvert I'\rvert/2$	159501	7002
ID	$Const_3$	5	67032	8043
	PD_3	[min-depth, 0.4$\lvert A\rvert$, $\lvert A\rvert$]	54321	7400
CSE			52646	8876

Table 7.1: Application of different methods on IRBC algorithm

these three approaches gives us 20% less number of expanded states but consumes in 2.5 more runtime than CSE algorithm.

In ID algorithm the best technique how to change max-depth is PD method. In Iterative Deepening, we approach the optimal solution of I_i when max-depth= $|A|$, where A is a set of possible actions. As it was mentioned, we proposed an algorithm which finds the minimal depth $\textit{min-depth}_{I_i}$ in order to reach some reward in AG. Therefore, for PD $\textit{list} = [a_1, a_2..a_n]$, where $a_{i+1} > a_i,\ a_1 = \textit{min-depth},\ a_n = |A|$. If I_i is satisfiable in m^{th} iteration, then $\textit{max-depth}_{I_i} = \textit{list}(m)$. PD_3 expands more states than IRBC, but it finds the solution almost as fast as CSE.

In conclusion for further investigating and comparing it with CSE algorithm, we will consider four highlighted methods in Table 7.1.

7.3 Scalability

In this section, we introduce the runtime and the number of expanded states for different techniques. As it is mentioned in the section above PD is the best approach to set up the tradeoff parameter between time and number of expanded states. In previous experiments elements of *list* in PD was chosen randomly. We ran more experiments on different networks with different *min-utility* in order to gain information about i) time of solving

game optimal with this *min-utility* ii) the probability of returning to the information set after the first iteration. With this information, we found the best coefficient for *list* in PD, which reduces an overall computational time for the game solution. Therefore, for IRBC algorithm and PD method $list = [0.74, 0.43, 0]$. For ID algorithm, it was difficult to set up *list* uniformly, because the different network has a different number of actions, and coefficients of *list* mostly depend on this number. Empirically the best *list* for ID algorithm is $list = [\textit{min-depth}, 0.52N, N]$.

In Figure 7.1 there is comparison of a number of expanded states of iterative algorithms for chain network and in Figure 7.2 the average runtime of these algorithms with five runs. We increase the number of host types T, the number of hosts n and number of honeypots k. The missing data demonstrates that the computer was not able to solve it due to small memory limit. From the results, it can be seen that in the majority of the networks iterative methods expand fewer states than CSE algorithm. As it was expected IRBC algorithm has better pruning characteristics than ID algorithm. HPD (H+PD) method applied to IRBC algorithm expands the least number of states, in average 23-35% less than CSE. But the time of iterative algorithms reaches twice more than in CSE algorithm, even though in some networks they show better computational time than CSE. The experiments were held for business network and reached similar results as for chain network: HPD technique appeared to be the most efficient.

7.4 Results

In conclusion, we introduce some techniques of setting up the tradeoff parameter and modification of iterative algorithms how to reduce the number of expanded states. Iterative Reward Bound algorithm with pruning technique HPD presents the best solution, reducing 30% of the state. We have expected that decreasing number of expanded states we would decrease overall computational time. Unfortunately, experiments show that iterative approaches are slower than CSE algorithm. The bottle-neck of the iterative algorithms is that each IS in every iteration should be solved from initial state until one of the states is pruned, or IS is solved optimally. The solution how to reduce the time is for every IS to remember the set of states S which has been pruned in the previous iteration and if IS is again satisfiable, then to begin computation from the set states S. This approach has some negative aspects. Firstly for a larger network it will fall on memory

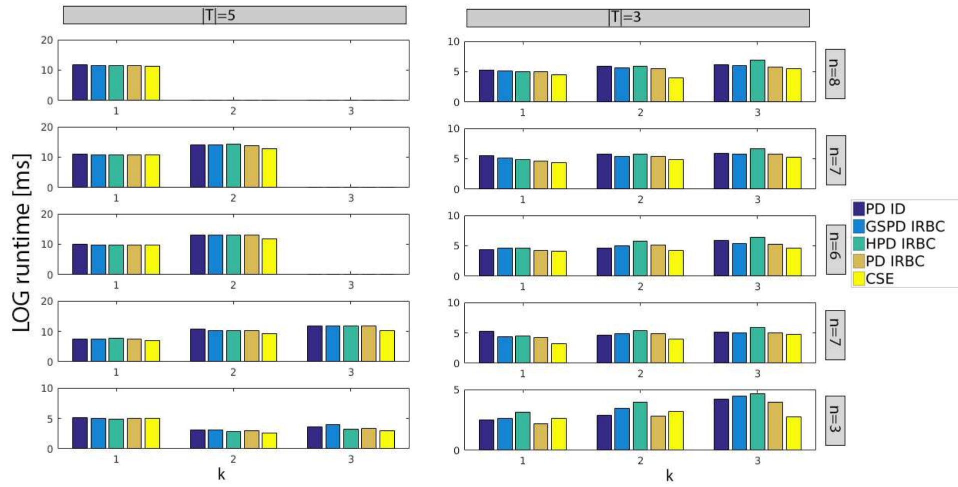


Figure 7.1: Comparison of approximation scalability for for chain network with t type hosts, n hosts and k honeypots.

limit. Secondly, after each iteration LP demands of the CURB set for each information set, meaning that attack policies should be recomputed for each IS.

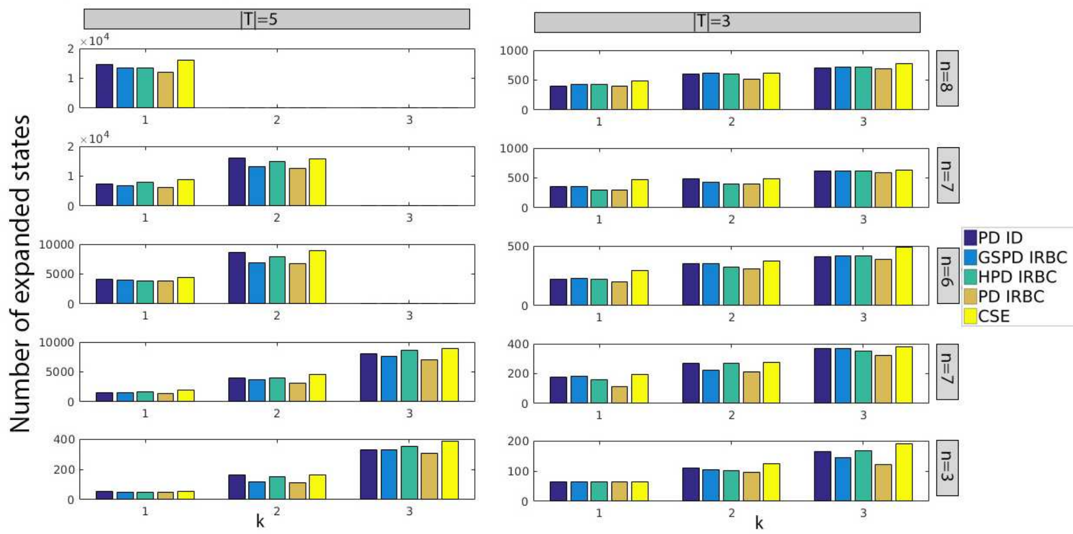


Figure 7.2: Comparison of number of expanded states of different approaches for chain network with t type hosts, n hosts and k honeypots.

Chapter 8

Conclusion

We study a network security game, where the attacker has imperfect information about the network. We continue investigating the previous work of finding the optimal SSE strategy profile. In this work, we try to find a tradeoff parameter for CSE algorithm to reduce the overall computational time of the game. We examine two iterative algorithms which avoid solving the game completely and still finding the optimal solution. We analyze different techniques for the tradeoff parameter and apply them to iterative algorithms. Some of them are efficient and reduce the game complexity by 30%.

Unfortunately, we have not decreased overall computational time. The problem is that we recompute attack policies in the attack graphs from the beginning in every iteration, even if in the previous iteration they were partially solved. If we remember the structure of the attack graph in every iteration, we expect to reduce time also by 30%. But it is insufficient to approach due to the memory limit.

In further research, we plan to focus on the quality of the attacker's CURB. We suggest not to solve it optimally and neglect some attack policies, to find the compromise between the runtime and optimal solution.

Personally, I have discovered a study called Game Theory and gained knowledge in the network security. I am interested in the exploration of this problem and will continue to work on it.

Bibliography

Bosansky, B. a Cermak, J. (2015), Sequence-form algorithm for computing stackelberg equilibria in extensive-form games, *in* 'Proceedings of the Twenty-ninth Aaai Conference on Artificial Intelligence', AAAI Press.

Cassandra, A., Littman, M. L. a Zhang, N. L. (1997), Incremental pruning: A simple, fast, exact method for partially observable markov decision processes, *in* 'Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence', Morgan Kaufmann Publishers Inc., pp. 54–61.

Daintith, J. a Wright, E. (2008), *A dictionary of computing*, Oxford University Press, Inc.

Durkota, K., Lisỳ, V., Bošanskỳ, B. a Kiekintveld, C. (2015*a*), Approximate solutions for attack graph games with imperfect information, *in* 'Decision and Game Theory for Security', Springer, pp. 228–249.

Durkota, K., Lisỳ, V., Bošanskỳ, B. a Kiekintveld, C. (2015*b*), Optimal network security hardening using attack graph games, *in* 'Proceedings of IJCAI', pp. 7–14.

Fudenberg, D. a Tirole, J. (1991), 'Game theory, 1991', *Cambridge, Massachusetts* **393**.

Noel, S. a Jajodia, S. (2004), Managing attack graph complexity through visual hierarchical aggregation, *in* 'Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security', ACM, pp. 109–118.

Osborne, M. J. (2004), *An introduction to game theory*, Vol. 3, Oxford University Press New York.

Qian, Y., Tipper, D., Krishnamurthy, P. a Joshi, J. (2010), *Information assurance: dependability and security in networked systems*, Morgan Kaufmann.

Yin, Z., Korzhyk, D., Kiekintveld, C., Conitzer, V. a Tambe, M. (2010), Stackelberg vs. nash in security games: Interchangeability, equivalence, and uniqueness, *in* 'Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1', International Foundation for Autonomous Agents and Multiagent Systems, pp. 1139–1146.