TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science
IAPB02/09

# Usage of Statistical Data in Rational Attack Feasibility Analysis

Bachelor's Thesis

*Student:* Žuljen Dedegkajev
*Student Code:* 093942IAPB
*Supervisor:* Dr. Aleksandr Lenin, Ph.D

TALLINN
2017

Tallinna Tehnikaülikool
Infotehnoloogia teaduskond
Tarkvarateaduse instituut
IAPB02/09

# Statistiliste Andmete Kasutamine Ratsionaalsete Rünnete Teostuvuse Analüüsimises

Bakalaureusetöö

*Tudeng:* Žuljen Dedegkajev
*Tudengikood:* 093942IAPB
*Juhendaja:* Dr. Aleksandr Lenin, Ph.D

TALLINN
2017

# Declaration of Authorship

Hereby I declare that this master thesis, my original investigation and achievement, submitted for the master degree at Tallinn University of Technology has not been submitted before for any degree or examination.

| | |
|---|---|
| Date | Author's signature |

# Abstract

Existing methods of security analysis either do not provide full overview of the situation or it is too laborious for analysis to apply them in practice. Often researchers seek for the answer how to automate this process. Attack tree analysis is a prominent security modeling and analysis method has gained importance over past decades. There are methods of automatic generation of large attack trees and security analysis, but in practice there is no reliable method of assigning quantitative annotations to the leaves of attack trees. This process still remains unautomated and requires consideration by experts in this field. Often in practice quantitative data for intermediate nodes is the only one available and can be harvested from various information sources. The main goal of this thesis is to create computational method, which will allow automatically populate the attack tree threat model, attain quantitative annotations on the attack tree leaves, taking into account available statistical data.

The thesis is in English and contains 46 pages of text, 3 chapters, 9 figures, 1 table.

# Annotatsioon

**Statistiliste Andmete Kasutamine Ratsionaalsete Rünnete Teostuvuse Analüüsimises**

Olemasolevad turvaanalüüsi meetodid ei anna situatsioonist kogu ülevaadet või on praktikas nende rakendamine analüüsiks liiga töömahukas. Uurijad otsivad tihti viise selle protsessi automatiseerimiseks. Ründepuudele põhinev analüüs on põhiline turvamodelleerimise ja analüüsi meetod, mille tähtsus on viimastel aastakümnetel tõusnud. Suurte ründepuude automaatseks genereerimiseks ja turvaanalüüsiks leidub meetodeid, kuid praktikas ei leidu usaldusväärset viisi ründepuu tippudele kvantitatiivseid annotatsioone määrata. See tegevus on siiani automatiseerimata ning nõuab ala asjatundjate tähelepanu. Sageli on vahesõlmede jaoks kvantitatiivsete andmete kogumine ainus võimalus, mida saab erinevatest allikatest hankida. Selle lõputöö põhiline eesmärk on luua arvutusmeetod, mis võimaldab olemasolevaid statistilisi andmeid arvesse võttes automaatselt täita ründepuu ohumudeli ning leida puu tippudele kvantitatiivsed annotatsioonid.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 46 leheküljel, 3 peatükki, 9 joonist, 1 tabel.

# Contents

# List of Figures

# List of Tables

# Introduction

## Real World Problem

The increasing amount of cyber accidents from year to year clearly shows the importance of information security. Its primary goal is to ensure that assets are sufficiently protected and deployed security measures provide the most efficient protection possible at the moment. Risk analysis is made in order to comprehend various system threats and effectively defend it from cyber attacks. Security measures cost money and the main goal of planning is to wisely invest funds into security. Taking into consideration current security risks, priorities and budget it is important to find a set of the most efficient security measures that can ensure reasonable security level with minimal investments.

There exist a lot of different categories of rational attackers starting from script-kiddies who have few skills and resources and who launch ready-made malware attacking mainly for fame ending with high skilled attackers who attack for profit. There also exist types of irrational attackers who attack for religious and belief reasons or for sake of revenge. Their attacks are almost impossible to predict, because they are rarely guided by logic [10, 9, 11]. From the point of view of operational security, damage induced by script-kiddies attacks belong to the category of residual risks. In practice, enterprises most of all are concerned with targeted attacks of high skilled attackers, who exactly know whom they are going to attack and conduct thorough analysis of attack feasibility and consider all potential risks. Damage from such attacks hundred or even thousand times exceeds the damage made by script-kiddies attacks. Therefore, from the point of operational security, enterprises primarily intend to protect themselves from targeted profit-oriented attacks in the first place.

In order to determine system's attack resistance level to targeted profit-oriented attacks, analysis of attack viability must be conducted, similar to the one that is carried out by the attacker while making decision about attack viability. From profit-oriented attacker's point of view, it is viable to attack if profit exceeds expenses. If expenses exceed potential profit, then attackers are likely not to be interested in attacking the system. The doctoral dissertation [7] introduces a model and computational methods in order to conduct feasi-

bility analysis of targeted rational profit-oriented attacks. This model uses attack tree as a threat model.

Typically, during attack tree analysis elementary attacks are annotated with parameters and computational methods are used to perform an analysis. [5] Usually, such parameters or quantitative annotations are estimated by experts based on their personal experience, public statistical data and threat intelligence collected by private enterprises. Reliability and confidence level of data directly depends on the input data quality. Often expert opinions can vary and even contradict with each other. [2, 4, 13, 14] So it is reasonable to rely on statistical data if it is available.

The weakness of the of model [7] is an unrealistic assumption that quantitative annotations of elementary attack are available, which is not always true in practice. This thesis focuses on the problem when security analysts have partial knowledge about attack scenarios. In this thesis, model [7] is applied in the case when quantitative annotations for elementary attacks are unknown to an analyst, but some dependencies of elementary attacks are known instead. The goal of the current thesis is to find an efficient solution to attack-feasibility problem. Questions regarding data availability, quality and reliability, as well as their statistical characteristics and filtering are out of the scope of this thesis. This work incrementally builds upon model for rational attackers and therefore attacks of irrational attackers are out of the scope of this thesis.

This thesis is a part of ADTool development iteration, as the result of which its functionality will be extended. Namely, it will become possible to analyze attack trees using statistical data. This thesis is a first step in this research. The main goal if to create a model and a working prototype.

## Research Questions

Research questions are the following:

1. What type of the problem are we dealing with?
2. How can it be solved?
3. Which approach is the most efficient one?

## Methodology

In order to understand how to solve a given problem it is necessary to comprehend which type of problem we are faced with. For this reason a mathematical model of studied phenomena is

composed. Having a mathematical description of the problem we start to seek for possibilities to solve it by comparing different alternatives. We evaluate effectiveness of computational methods based on their performance. In order to collect empirical data about computational methods a program is written which implements found computational methods.

## Expected results

Expected results of the thesis are the following:

1. Formal model of the considered problem.
2. The most efficient computational method found empirically that solves a given problem.
3. Algorithm description, which corresponds to computational method described later in thesis and algorithm implementation.

## Structure

State of the art outlines the relevant research in the area of quantitative attack tree analysis, establishes the context for this work, outlines the gap and justifies the relevance of the presented research. Chapter 1 discusses various representations of the considered problem – as an optimization task and as constraint satisfaction problem. Chapter 2 discusses the problems related to converting an XML formatted attack tree input file into the constraint satisfaction problem. Chapter 3 presents the implementation details and deployment guidelines. Open questions and future work sums up the contributions of this thesis, points out open questions and provides some pointers for future work.

# Abbreviations and Symbols

| Abbreviation | Definition |
|---|---|
| ADTOOL | Attack-Defense Tree Tool |
| ADTREE | Attack-Defence Tree |
| API | Application Programming Interface |
| CPU | Central Processing Unit |
| CSP | Constraint Satisfaction Problem |
| DAG | Directed Acyclic Graph |
| GLPK | GNU Linear Programming Kit |
| GUI | Graphical User Interface |
| JAVA SE | Java Platform, Standard Edition |
| JNI | Java Native Interface |
| JVM | Java Virtual Machine |
| LP | Linear Programming |
| MILP | Mixed-integer Linear Program |
| MIQCP | Mixed-integer Quadratically Constrained Programming |
| MIQP | Mixed-integer Quadratic Programming |
| MIP | Mixed Integer Programming |
| MS | Member States of the European Union |
| QCP | Quadratically Constrained Programming |
| QP | Quadratic Programming |
| RFID | Radio-frequency Identification |
| SLP | Sequential Linear Programming |
| SQP | Sequential Quadratic Programming |
| XML | Extensible Markup Language |
| XSD | XML Schema Definition |

| Symbol | Definition |
|---|---|
| $\mathbb{R}_{\geqslant 0}$ | A set of non-negative rational numbers |

# State of the Art

## Attack Trees

Attack tree is a directed acyclic graph (DAG). Such graph consists of nodes that are connected with edges. Attack tree compared to graph definition defines two additional types of nodes: root node and leaf node. Root node is a node that doesn't have parents. In attack tree, there can only be one root node. Leaf node is a node that doesn't have successors. Attack tree is a schematic representation of all possible attacks in the system. They are used for risk analysis, determination of system threats and security attributes. An attack tree is usually created by a team of security analysts and experts.

In 1991, Weiss [15] introduced threat logic trees as the first graphical attack modeling technique. He proposed a first simple procedure for quantification of attack trees, which was based on a bottom-up algorithm. In this algorithm, values are provided for all leaf nodes and the tree is traversed from the leaves towards the root in order to compute values of the refined nodes. Depending on the type of verifying, the various functional operators are used to combine the values of the children. This procedure allows to analyze simple aspects such as the costs of performing attack, attack time or the necessary skill level. The existing literature on attack modeling approaches that are based on directed acyclic graphs is quite rich. Nowadays, there exist more than 30 different approaches for analysis of attack and defense scenarios. Many authors treat how to add different kinds of values to attack tree nodes [1]. In Schneiers terms [12] those values are called attributes. In 1999, he proposed how to analyze the costs and the success probability of an attack with the help of attack trees. Attributes provide a powerful analysis tool for vulnerability scenarios. Many other researchers have continued to develop analysis models proposing extensions to attack trees and attributes, as well as describing case studies. In 2005, Mauw and Oostdijk [8] formalized the basic concepts of attack trees that were informally introduced by Schneier in 1999. They provided a denotational semantics using a mathematical framework, based on a mapping to attack suites, which abstracts from the internal structure of an attack tree. Researchers specified tree transformation between attack trees consistent with their framework, defined attributes on attack trees, that guarantees compatibility with underlying

semantics, and discussed under which conditions they can be synthesized bottom-up. They also studied the projection of an attack tree. Attack tree analysis provide a powerful analysis tool for vulnerability scenarios. They help us estimate which attacks may happen with a high probability and which countermeasures should be applied. However, to get useful insights from the analysis, it is necessary to have accurate values associated with all the nodes of an ADTree [1]. The importance of our problem is that public statistical data is usually not available on all nodes of attack tree, because there is no official information available on costs of elementary attacks. However, it is still possible to determine attack cost, which is illustrated by the recent survey conducted in the Netherlands found the average theft of a laptop cost to attacker between 200-500€ [3].

## Attack-Defense Trees

One of the formal approaches to assess a systems security is the attackdefense tree (ADTree) methodology [1]. The theoretical aspects of the ADTree methodology were developed by Kordy et al. ADTrees are an extension of attack trees with defense nodes. An ADTree is a node-labeled rooted tree describing the measures an attacker might take in order to attack a system and the defenses a defender can employ to protect the system. ADTrees allow the system modeler to repeatedly interleave attack and defense components. ADTrees can be used as part of threat and risk analysis for system development and maintenance. Bagnato et al. have conducted a practical case study with ADTree for Radio-Frequency Identification (RFID) system for goods management system for a warehouse [1]. They focused on the use of attributes in ADTrees demonstrating how an attack tree can be parametrized with different kinds of values, such as costs of performing an attack, profit the attacker will receive should the attack succeed, and performing quantitative analysis.

## Statistical data and threat intelligence

Attacks against information systems are a growing menace in the European Union and globally. There are few statistics available on attacks against information systems. Many police forces do not keep statistics on the computer systems involved in these and other attacks [21]. However, the European Union has started to address this issue by collecting some figures on attacks against information systems. In accordance with Directive 2013/40/EU [16] of the European Parliament and of the Council on attacks against information systems Member States (MS) shall take the necessary measures against the illegal access to information systems, illegal system and data interference, and illegal interception. European Commission publishes statistical reports based on those data and submits to the competent specialized

Union agencies and bodies. Besides public sources, large companies maintain their own databases containing threat intelligence, which is highly sensitive information. Many companies are not willing to report cases of attacks to avoid negative impact on reputation and exposure to future attacks. Global surveys find that 15% of businesses say they have faced a cyber-attack in the past year.

## Conclusion

Many analysis techniques exist out there and all of them rely on quantitative data supplied to the attack tree leaves. The approach of the incorporating available statistical data and interleaving it with expert estimations hasn't been extensively studied yet. This thesis aims to fill this gap by applying available statistical data about the cost of intermediate nodes in attack trees in order to determine feasibility of attack. This thesis is based on the improved failure-free model created by Lenin A. in 2015 [7]. This security analysis model considers a broader scope of attackers. It allows the attacker to repeat elementary attacks if they fail and to continue attacking when launched attacks are detected. In the model, there exist optimal attacking strategies, which mean that the next tried sub-attack does not depend on the results of the previously tried attacks. Compared to the previous models and methods, the improved failure-free model proposes a more reliable model and more efficient computational methods for calculating upper bounds of the expected outcome of rational attackers, taking into accounts fully adaptive adversarial setting in which the attacker is able to run atomic sub-attacks in arbitrary order. The new model is particularly useful for developing reliable engineering methods in information security.

# Chapter 1

# Attack feasibility determination

This chapter considers the question which type of problem we are dealing with. The goal is to obtain a formal description of the given problem in order to solve it. In this chapter the problem of determination of feasibility of rational attacks is equivalent to Constraint Satisfaction Problem(CSP) will be demonstrated.

For attack feasibility estimation attacker expenses are compared to profit. Attack is feasible if profit exceeds expenses [7]. From security point of view we are interested to demonstrate that attacks in our model are unviable. E.g. if attacker's minimal expenses are 1000€ and profit gained from the attack is 500€, then a conclusion can be made that our system is secure against targeted profit-oriented rational attacks, because attacker will suffer losses. But such logic leaves the opportunity for false-positive results when computations show that the system is secure, but in reality it is not true. Thus the results of these computational methods are unreliable. False-positive results may happen, because cheaper attack paths exist in the analyzed system [7].

In order to avoid false-positive results, lower bound of the attack expenses must be considered. False-positive results occur when the analysis tells us that the system is secure, but in reality it is insecure. This may happen if the computational methods calculate the lower bond of attack expenses precisely. The result of such analysis cannot ensure that there does not exist a cheaper way to attack the system. In order to give such guarantee the lower bound of attack expenses must be calculated. Such result guarantees that there are no cheaper ways to attack the system than the calculated value, thereby it eliminates false-positive results.

Let us examine the attack tree shown in Fig. 1.1. The costs of the elementary attacks A, B, C, D are unknown, but partial knowledge on the parts of attack scenario is available. In attack tree analysis elementary attacks are annotated with quantitative annotations chosen for analysis. Such attributes can be e.g. attack cost, probability of attack's success, attacker

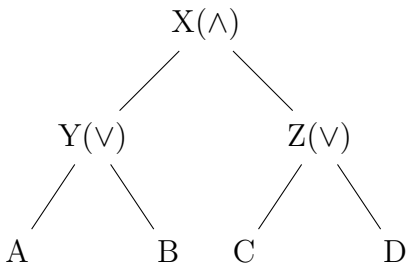



FIGURE 1.1: An example of the attack tree

skill level and experience etc. Restrictions in attack tree nodes can be the following: equality (=) and inequalities ($\leqslant, \geqslant$). Equality denotes that the subtree cost must be exactly equal to value from initial conditions. Inequalities set lower and upper bounds for the cost of attack subtree.

## 1.1 ATTACK TREE AS A FUNCTION

Attack tree analysis relies on the concept of attribute domains [8]. An attribute domain is a triplet (D, $\wedge$, $\vee$), where D denotes domain of the quantitative value (e.g. $\mathbb{R}_{\geqslant 0}$ - set of nonnegative numbers, attack cost cannot be represented by negative value), $\wedge$ and $\vee$ are predicates to be applied in conjunctive and disjunctive operators respectively. For every tree node attribute domain sets input function of this node. Attribute domain (D = $\mathbb{R}_{\geqslant 0}$ , +, min) corresponds to the minimal cost of the attack tree it is applied to. The min and sum functions go in line with the rationality assumption. In $\wedge$-nodes the attacker executes all the possible attacks and in $\vee$-nodes execute only the cheapest attack.

Every attack tree node represents a function of attack tree inputs. For instance, given attribute domain ($\mathbb{R}_{\geqslant 0}$ , +, min) an attack tree shown in Fig. 1.1 represents function $f(A, B, C, D) = \min(A, B) + \min(C, D)$.

Given a minimum cost domain the function represented by an attack tree is either a min function or an affine combination of min functions, therefore the function is nonlinear, nonconvex, not twice continuously differentiable, but has subgradients. Attack feasibility can be determined in a variety of ways.

One way is to find the minimal value of the function satisfying given constraints and compare this minimal value to the attacker profit. This means that a nonlinear optimization task needs to be solved and a nonlinear nonconvex objective function must be minimized

over the feasible set defined by the set of nonlinear constraints.

$$
\begin{aligned}
\text{minimize} \quad & f(x_1, \ldots, x_n) \qquad \text{subject to} \\
& g_1(x_1, \ldots, x_n) = 0 \\
& \qquad \vdots \\
& g_k(x_1, \ldots, x_n) = 0 \\
& h_1(x_1, \ldots, x_n) \leqslant 0 \\
& \qquad \vdots \\
& h_m(x_1, \ldots, x_n) \leqslant 0
\end{aligned}
$$

Such type of problems is constrained nonlinear optimization problem. Quite often these problems are not solvable exactly and various approximation techniques, such as meta-heuristics, exist and provide approximate solution. This thesis looks into possible ways to simplify this task by means of mixed-integer linear program (MILP) relaxation. MILP optimizes linear objective function subject to nonlinear set of constraints and is therefore expected to be more robust compared to solving nonlinear optimization problem. Such a setting however requires to investigate if the structure of the problem can be exploited and if the objective function can be linearized.

The second way to calculate the minimal cost is to represent the task in the form of a constraint satisfaction problem (CSP). CSP solving is a method present in almost every optimization solver. CSP is solved by a search algorithm and one wishes to determine the existence of a solution. In order to determine feasibility of attack additional constraint needs to be added to the set of constraints – the constraint which requires the minimal cost to be less or equal to attacker profit. If CSP finds a solution then attacking is feasible following the rationality assumption. If CSP is unsolvable then the are no profitable ways to attack and attacking is infeasible.

Consider an example in Fig. 1.2. The corresponding optimization task is shown in optimization task (1.1).

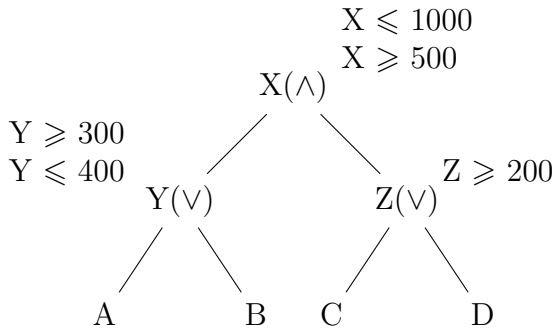



FIGURE 1.2: Attack tree with limitations

$$
\begin{aligned}
&\text{Minimize: } \min(A,B)+\min(C,D)\\
&\text{Subject to:}\\
&\qquad \min(A,B) \geqslant 300\\
&\qquad \min(A,B) \leqslant 400\\
&\qquad \min(C,D) \geqslant 200\\
&\qquad \min(A,B)+\min(C,D) \geqslant 500\\
&\qquad \min(A,B)+\min(C,D) \leqslant 1000
\end{aligned}
\tag{1.1}
$$

## 1.2 MILP FORMULATION

MILP optimizes linear objective function subject to nonlinear constraints. This thesis investigates possibilities to exploit the structure of an attack tree to encode the objective function in linear form. It is possible to assign a variable to every node in an attack tree and represent the layered tree structure of an attack tree.

MILP generation is a two-step process.

1. Assign variables shown in Fig. 1.3.
2. Generate MILP as shown in optimization task (1.2).

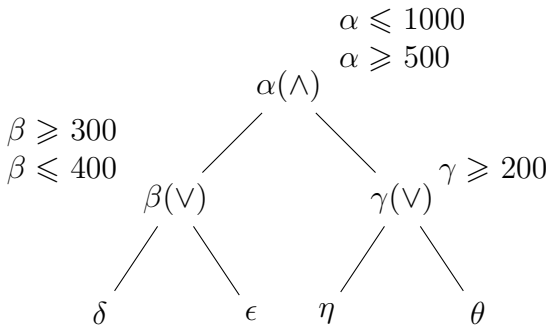



FIGURE 1.3: Attack tree with limitations

Minimize: $\alpha$

Subject to:

$$\begin{aligned}
&\alpha = \beta + \gamma \\
&\beta = \min(\delta, \epsilon) \\
&\gamma = \min(\eta, \theta) \\
&\alpha \geqslant 500 \\
&\alpha \leqslant 1000 \\
&\beta \geqslant 300 \\
&\beta \leqslant 400 \\
&\gamma \geqslant 200 \\
&\delta \geqslant 0 \\
&\epsilon \geqslant 0 \\
&\eta \geqslant 0 \\
&\theta \geqslant 0
\end{aligned} \tag{1.2}$$

However this problem formulation turned out to be incorrect as the sum of costs under every $\wedge$ node is constant and therefore no optimization is performed there. The solver falls back to solving CSP under $\wedge$ nodes. The CSP formulation of the problem is described below.

## 1.3 CSP FORMULATION

Similarly to MILP, all the nodes in attack tree in Fig. 1.3 are assigned with optimization variables. CSP has no objective function and is defined as a set of objects and a set of

constraints. CSP will now look as follows:

$$
\begin{aligned}
&\text{Objects: } \alpha, \beta, \gamma, \delta, \epsilon, \eta, \theta \\
&\text{Subject to:} \\
&\qquad \alpha = \beta + \gamma \\
&\qquad \beta = \min(\delta, \epsilon) \\
&\qquad \gamma = \min(\eta, \theta) \\
&\qquad \alpha \geqslant 500 \\
&\qquad \alpha \leqslant 1000 \\
&\qquad \beta \geqslant 300 \\
&\qquad \beta \leqslant 400 \\
&\qquad \gamma \geqslant 200 \\
&\qquad \delta \geqslant 0 \\
&\qquad \epsilon \geqslant 0 \\
&\qquad \eta \geqslant 0 \\
&\qquad \theta \geqslant 0
\end{aligned}
\tag{1.3}
$$

## 1.4 Conclusion

This chapter examines formalization of feasibility of rational attacks. In order to make a decision about attack feasibility, attack cost lower bound needs to be compared to the attacker profit. This can be achieved by formalizing the feasibility problem as a CSP, which contains a constraint requiring the cost of the root node not to exceed the attackers profit. If such a CSP is solvable then there exist profitable attack vectors and therefore rational attackers have incentive to attack and attacks against the considered system are feasible. If CSP is unsolvable it is infeasible to attack from rational attackers point of view.

# Chapter 2

# Generating CSP for a given attack tree

This chapter considers the procedure of creation of a CSP for a given attack tree. In the result an algorithm that calculates the minimum cost of the attack is created and is written in the form of a pseudocode.

## 2.1 XML representation of an attack tree

In order to generate CSP for a given attack tree, it should be encoded in suitable format at first. The basic format of the attack tree encoding is taken from the AttackDefense Tree Tool (ADTool) [6]. This tool allows users to model and display attack–defense scenarios, through the use of attack–defense trees (ADTrees) or an alternative term-based representation of ADTrees called attack–defense terms (ADTerms). Furthermore, the ADTool allows to perform quantitative analysis on ADTrees/ADTerms allowing user to answer questions such as: What is the cost of an attack, what is the minimal skill level required for an attacker to succeed, how long does it take to implement all necessary defenses or who is the winner of the considered attackdefense scenario and many others.

An XML Schema Definition (XSD) in Listing A.1 describes the format of the attack tree in the form of XML file [6]. Only XML files conforming with this schema are valid and can be imported by the application. Let us now take a closer look at XSD schema in Listing A.1. The XML file conforming with adtree.xsd contains exactly one "adtree" element representing the entire tree (lines 3–9 in Listing A.1). Each node of an ADTree is represented by a "node" element (lines 10–17 in Listing A.1), which can have exactly one sub-element "label", corresponding to the label of the node and a finite number of "node" sub-elements, corresponding to the nodes children, including countering nodes. There are

two attributes that characterize "node" elements: "refinement" attribute and "switchRole" attribute. "refinement" attribute represents the refinement of the node in the ADTree. This attribute has two possible values: "disjunctive" and "conjunctive" (lines 24–29 in A.1). The default value of the "refinement" attribute is "disjunctive", thus only conjunctive nodes require an explicit specification of the value for the "refinement" attribute. The values of the element "label" (lines 18–23 in A.1) are strings that can be composed of the following characters: numbers, capital and small letters of the English alphabet, space, horizontal tab, enter, question mark, exclamation mark, dash, underscore and dot.

But the following XSD schema doesn't support constraints, therefore it was necessary to extend its format (shown in Listing B.1) by adding an additional optional sub-element "constraint" (lines 36–41 in B.1) to "node" element that characterizes it by adding a constraint. Each "node" element can have zero or more "constraint" elements and each of them adds its own restriction. The "constraint" attribute has three possible values: "LE" – lesser or equal, "GE" – greater or equal and "EQ" – equal, all written with capital letters of the English alphabet, followed by one or more spaces and a double value. Let us consider an attack tree exported in XML format in Listing 2.1. In XML file "constraint" element is encoded inside a special tag on lines 5, 8, 9 and 13 in Listing 2.1. E.g. node A has the <constraint>LE 100</constraint> constraint specified, which means that optimization variable in node A should be lesser or equal to 100.

```
1 <?xml version='1.0'?>
2 <adtree>
3   <node refinement="conjunctive">
4     <label>A</label>
5     <constraint>LE 100</constraint>
6     <node refinement="conjunctive">
7       <label>B</label>
8       <constraint>GE 0</constraint>
9       <constraint>LE 50</constraint>
10    </node>
11    <node refinement="conjunctive">
12      <label>C</label>
13      <constraint>EQ 50</constraint>
14    </node>
15  </node>
16 </adtree>
```

Listing 2.1: Example of attack tree exported into XML

## 2.2 Creation of a CSP

Let us consider an attack tree with N nodes encoded as an XML file and an attribute domain ($\mathbb{R}_{\geqslant 0}$, +, min). Some nodes of this attack tree have specified constraints. This thesis considers an example where the minimal cost of the attack is calculated. In order to calculate the minimal cost of the attack, CSP that corresponds to the attack tree needs to be solved. Attack tree needs to be represented in the form of a set of CSP constraints. CSP algorithm recursively runs through each node of the attack tree and creates an optimization variable for the corresponding node. If the given node has constraints, they are added to the CSP. Let us consider the following example: given a variable X and constraint EQ 100, then the expression X = 100 is added to a CSP. If a node has several constraints, then several corresponding expressions are added to a CSP. Also, it is necessary to reflect the attack tree structure in a CSP, namely the relation of each node with its children. Let us consider the following examples of attack trees: $x_1 = x_2 \wedge x_3$ and $y_1 = y_2 \wedge y_3$. For attack tree $x_1$ a CSP constraint will be written if the form of expression $x_1 = x_2 + x_3$ and for $y_1$ attack tree $y_1 = \min(y_2, y_3)$.

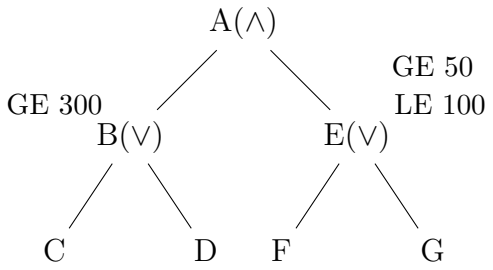



Figure 2.1: Attack tree traversal initial state

Let us demonstrate the above mentioned procedure by an example: Let us consider the attack tree in Fig. 2.1. The algorithm starts to traverse the attack tree in Fig. 2.1 starting from root node A. Since node A is a disjunctive node, the value of optimization variable $x_1$ refer to Fig. 2.2 in root node A is equal to the value of the minimum function of its children: $x_1 = x_2 + x_5$. However, since a node B is a conjunctive node, the value of optimization variable $x_2$ in node B is equal to the sum of its children $x_2 = x_3 + x_4$. The following constraint: $x_1 \geqslant 300$ is added to the optimization variable $x_2$. Since C and D are attack tree leaves, their corresponding optimization variables $x_3$ and $x_4$ are added constraints $x_3 \geqslant 0$ and $x_4 \geqslant 0$ respectively, since cost is a non-negative value. The process mentioned above is illustrated by Fig. 2.2.

Similarly to node B the value of optimization variable $x_5$ in node E is: $x_5 = x_6 + x_7$. The following constraints $x_5 \geqslant 50$ and $x_5 \leqslant 100$ are added to the optimization variable $x_5$.

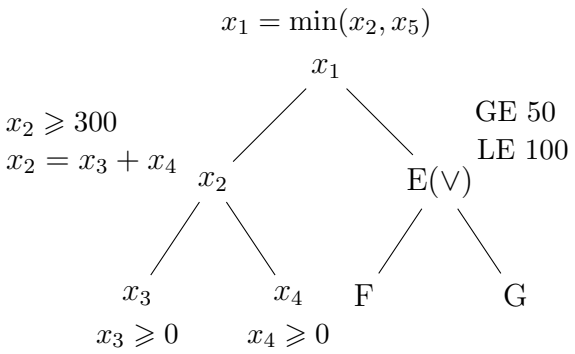



FIGURE 2.2: Attack tree traversal intermediate state

Optimization variables $x_6$ and $x_7$ in nodes F and G are added constraints $x_6 \geqslant 0$ and $x_7 \geqslant 0$ respectively. In result of this process the set of constraints shown in Fig. 2.3 is obtained.

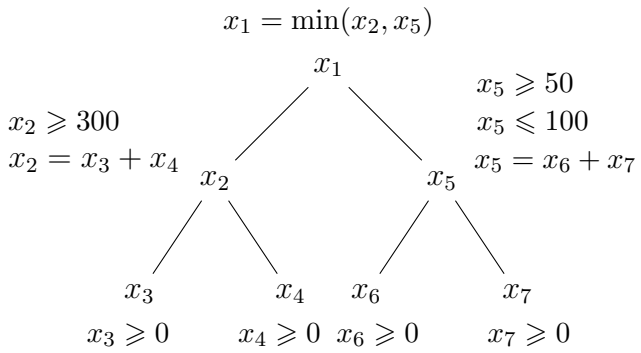



FIGURE 2.3: Attack tree traversal final state

Objects: $x_1, x_2, x_3, x_4, x_5, x_6, x_7$

Subject to:

$$
\begin{aligned}
& x_1 = \min(x_2, x_5) \\
& x_2 = x_3 + x_4 \\
& x_2 \geqslant 300 \\
& x_3 \geqslant 0 \\
& x_4 \geqslant 0 \\
& x_5 = x_6 + x_7 \\
& x_5 \geqslant 50 \\
& x_5 \leqslant 100 \\
& x_6 \geqslant 0 \\
& x_7 \geqslant 0
\end{aligned}
\tag{2.1}
$$

CSP pseudocode is presented below in Alg. 1.

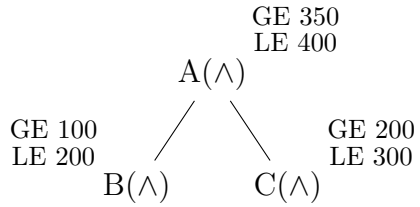
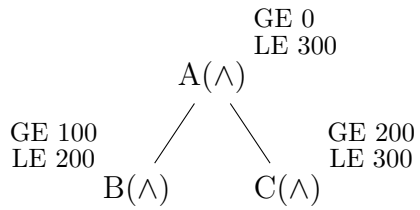
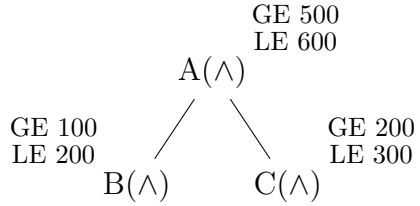
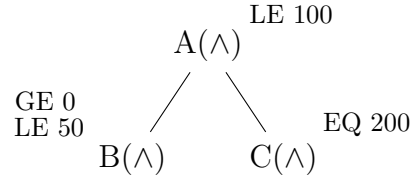
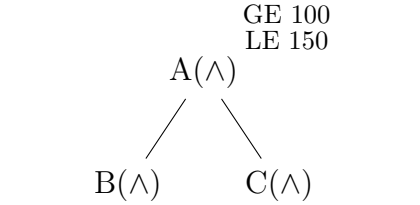
**Algorithm 1** Recursive composition of CSP from an attack tree

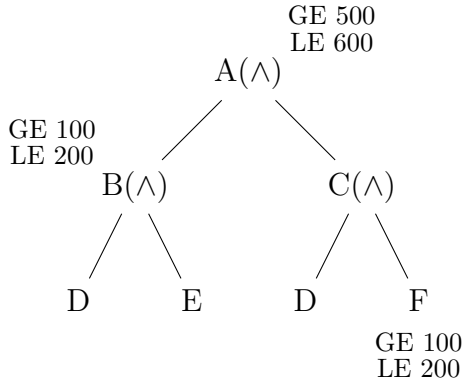
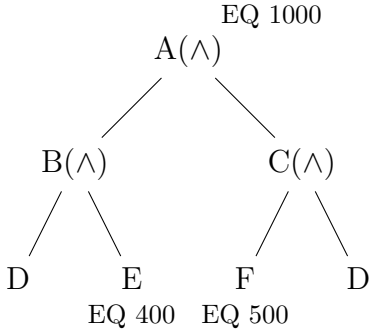
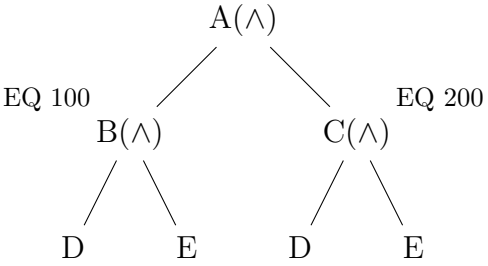
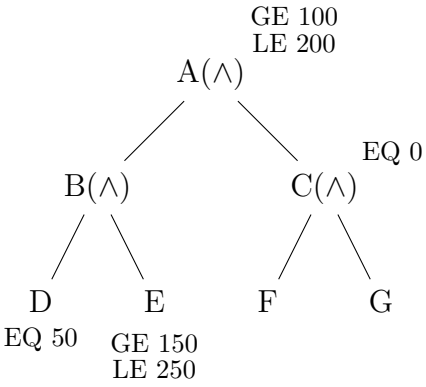
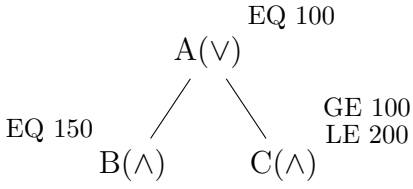
```
1: function createCSP(xmlNode node)
2:     var temp = createCSPVariable();
3:     List nodeConstraints = getNodeConstraints(node);
4:     for each constraint in nodeConstraints do
5:         var parsedConstraint = parseConstraintFromXml(constraint);
6:         addCspConstraint(temp, parsedConstraint.getOperator(),
7:                 parsedConstraint.getValue( ));
8:     end for
9:     if nodeIsLeaf(node) then
10:        addCspConstraint(temp, ≥, 0);
11:        return temp;
12:    end if
13:    List childrenNodes = getNodeChildren(node);
14:    List childrenVariables;
15:    for each childNode in childrenNodes do
16:        var temp2 = createCsp(childNode);
17:        childrenVariables.add(temp2);
18:    end for
19:    if nodeIsConjunctive(node) then
20:        addCspConstraint(temp, =, sum(childrenVariables));
21:    end if
22:    if nodeIsDisjunctive(node) then
23:        addCspConstraint(temp, =, min(childrenVariables));
24:    end if
25:    return temp;
26: end function
```

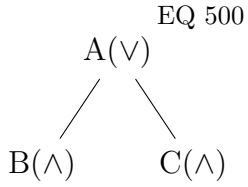
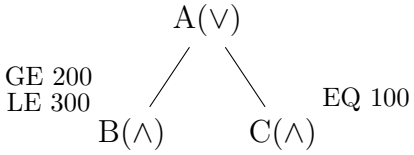
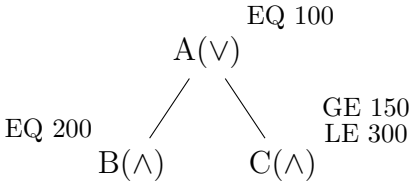
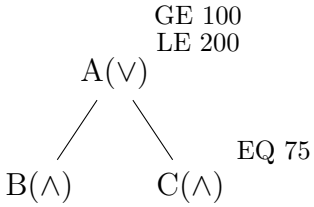
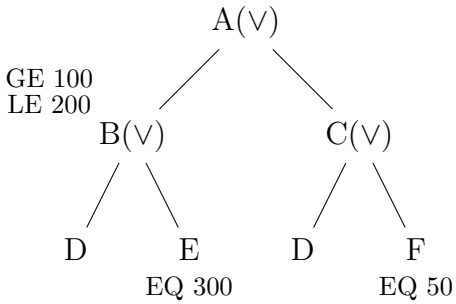
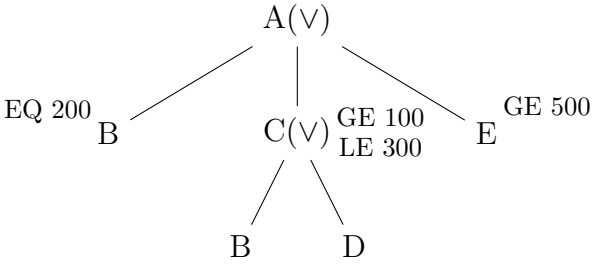
## 2.3 Unit Tests

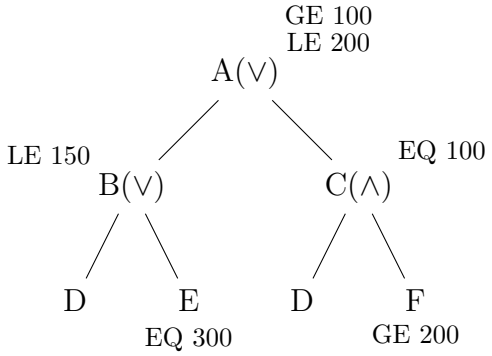
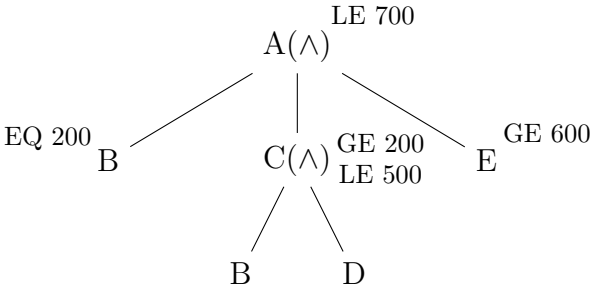
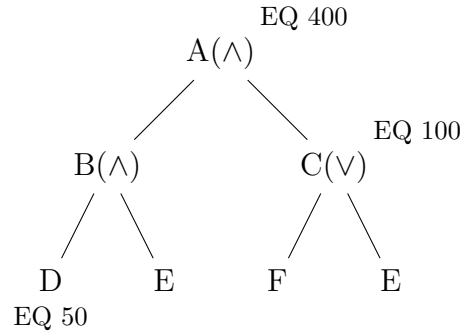
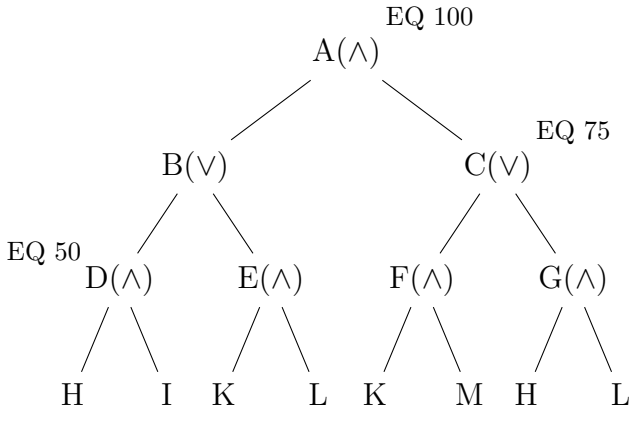
This chapter contains unit tests to ensure the validity of approach. These tests were created in a nonsystematic manner.

Table 2.1: Attack tree test cases

| Test nr | Input | Expected | Output |
|---|---|---|---|
| 1. |  | Solution exists | Solution exists |
| 2. |  | Solution exists | Solution exists |
| 3. |  | Solution exists | Solution exists |
| 4. |  | Solution infeasible | Solution infeasible |
| 5. |  | Solution exists | Solution exists |

| | | | |
|---|---|---|---|
| 6. |  | Solution exists | Solution exists |
| 7. |  | Solution exists | Solution exists |
| 8. |  | Solution infeasible | Solution infeasible |
| 9. |  | Solution exists | Solution exists |
| 10. |  | Solution exists | Solution exists |

| | | | |
|---|---|---|---|
| 11. |  | Solution exists | Solution exists |
| 12. |  | Solution exists | Solution exists |
| 13. |  | Solution infeasible | Solution infeasible |
| 14. |  | Solution infeasible | Solution infeasible |
| 15. |  | Solution exists | Solution exists |
| 16. |  | Solution exists | Solution exists |

| | | | |
|---|---|---|---|
| 17. |  | Solution infeasible | Solution infeasible |
| 18. |  | Solution infeasible | Solution infeasible |
| 19. |  | Solution exists | Solution exists |
| 20. |  | Solution exists | Solution exists |

## 2.4 Conclusion

In this chapter the process of drafting CSP for a random attack tree was considered. In the result the existing attack tree encoding implemented by ADTool was extended. Additionally,

the process of transforming a CSP encoded in ADTool Extensible Markup Language (XML) format into a CSP-native representation was developed and documented in pseudocode.

# Chapter 3

# Implementation

## 3.1 Solver

The variety of available optimization software able to solve CSP is rich. The main goal was to find a solver that could be used in the next iteration of ADTool and that could also meet the following conditions:

1. Ease of licensing. Freeware or free for academic use.
2. Developer–friendliness of Application Programming Interface (API).
3. Performance.
4. Tool support.
5. Well documented.

Considering the above was decided to choose between the most popular optimization software tools available: GLPK, Gurobi and CPLEX.

The GNU Linear Programming Kit (GLPK) is a free software package intended for solving large-scale linear programming (LP), mixed–integer programming (MIP), and other related problems [17]. It is a set of routines and organized in the form of a callable library. The package is part of the GNU Project and is released under the GNU General Public License. An independent project provides a Java-based interface to GLPK using Java Native Interface (JNI) which allows Java applications to call out to GLPK [18].

The Gurobi Optimizer is a commercial optimization solver distributed with a free academic license, among others. It is used for LP, quadratic programming (QP), quadratically constrained programming (QCP), mixed–integer linear programming (MILP), mixed–integer quadratic programming (MIQP), and mixed–integer quadratically constrained programming

(MIQCP) [19]. Gurobi has proven itself to be both robust and scalable, and is capable of solving problems involving millions of decision variables. Additionally, the power of the library is backed by a broad range of intuitive interfaces. The Gurobi Optimizer supports a variety of programming and modeling languages, including object-oriented interfaces for C++, Java, .NET, and Python.

CPLEX Optimizer is a commercial optimization software package developed by IBM and distributed with a free academic license, among others. It provides flexible, high-performance mathematical programming solvers for linear programming, mixed integer programming, quadratic programming, and quadratically constrained programming problems [20]. CPLEX delivers the power needed to solve very large, real-world optimization problems, as well as the speed. For free academic use the size of the optimization problem is limited to one thousand variables. Robust and reliable algorithms of CPLEX Optimizer are able to solve problems with millions of constraints and variables. Besides, its modeling layer provides flexible interfaces to the C++, C#, and Java languages, which gives developers a variety of ways to interact with it.

All of the above-mentioned optimization software tools support Java interface. This is important, since ADTool is written in Java programming language. Considering developer-friendliness, GLPK has the least friendly API among all, therefore the author decided to exclude GLPK from the list. E.g. unlike CPLEX and Gurobi, GLPK does not provide the possibility of adding constraints to the optimization model using provided API calls like min(), max(), abc(), which internally are represented as a set of mathematical constraints in the optimization task. Since CPLEX and Gurobi are developed by industry-leading companies, they both have great performance and tool support making my decision harder. Finally, the author decided to choose CPLEX over Gurobi, because CPLEX in my opinion is better documented, which made my programming task easier.

## 3.2 Architecture

Program architecture consists of two parts. First part is my contribution in the form of .jar executable archive. Second part consists of the CPLEX installation, which consists of cplex.jar archive and a set of dynamic system libraries. The whole algorithm of CPLEX is executed by the native libraries and cplex.jar is a wrapper, which provides the user with a set of API calls allowing him to execute native libraries functions.

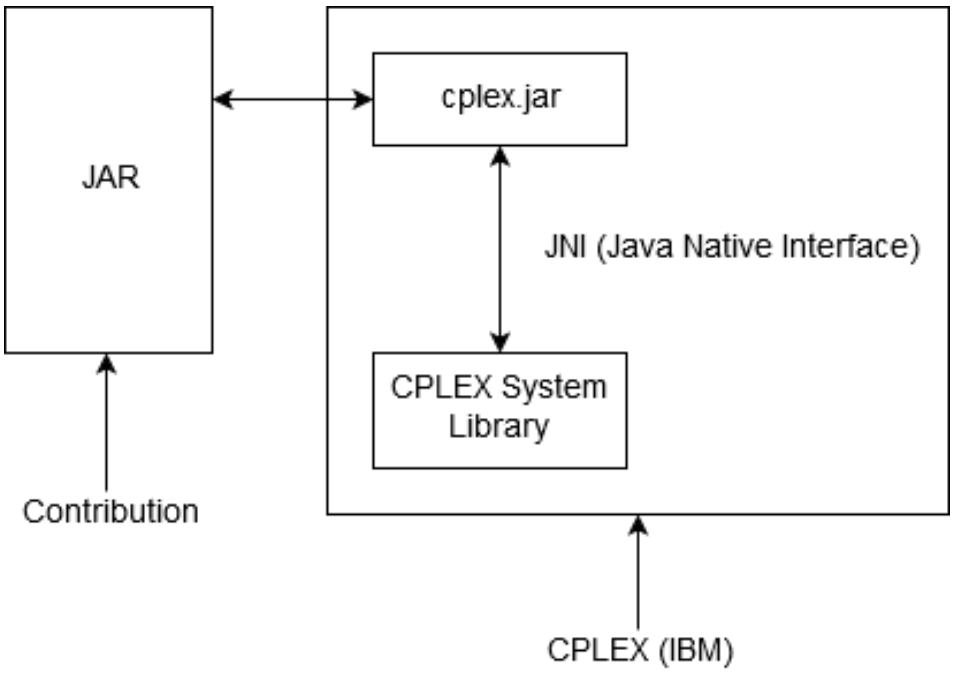



FIGURE 3.1: Program architecture

## 3.3 INTERFACE

One of the contributions of this thesis is a prototype front-end in JavaFX, which is a standard graphical user interface (GUI) library for Java Platform, Standard Edition (Java SE). The front-end consists of the main panel and its elements: textfield and textarea. In order to perform analysis, the user has to drag and drop one XML file into the textarea at a time. As soon as the XML file is dragged into the textarea, the analysis is executed automatically and the results are displayed to the user. If the XML file is valid, then the textfield will show the path to it. If the XML file is invalid, then the following message is displayed in the textfield: "Please select a valid XML file". An example input XML file is shown in Listing 3.1.

```
1 <?xml version='1.0'?>
2 <adtree>
3   <node refinement="conjunctive">
4      <label>A</label>
5      <constraint>GE 500</constraint>
6      <constraint>LE 600</constraint>
7      <node refinement="conjunctive">
8         <label>B</label>
9         <constraint>GE 100</constraint>
10         <constraint>LE 200</constraint>
11      </node>
12      <node refinement="conjunctive">
13         <label>C</label>
14         <constraint>GE 200</constraint>
15         <constraint>LE 300</constraint>
16      </node>
17   </node>
18 </adtree>
```

Listing 3.1: XML input file

The result output of program is shown in Fig. 3.2. On the top of the output screen is printed the optimization model describing the CPLEX object, which contains added constraints. Beneath it is displayed the solution status. Further below the list with attack tree nodes, their corresponding variables, attribute domains and optimal solutions is displayed.

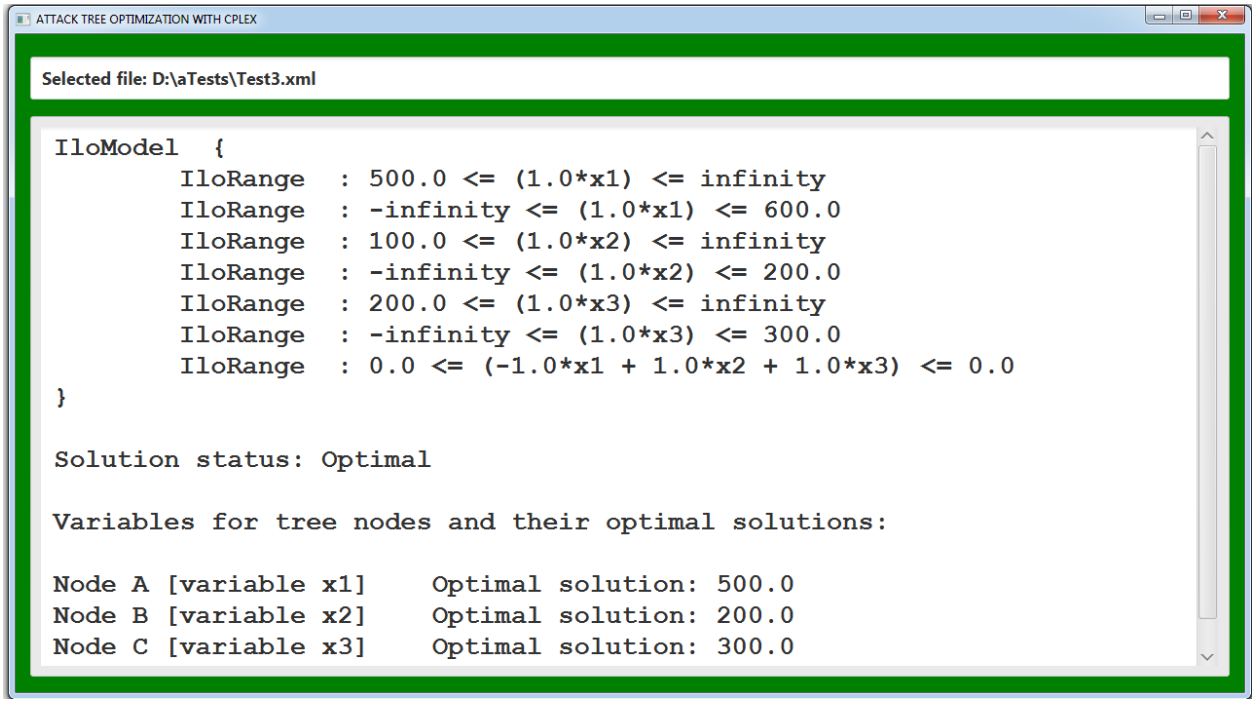



FIGURE 3.2: Program output

## 3.4 FUNCTIONALITY

In the beginning input XML file is passed to XML parser. XML parser XML parser converts an input attack tree with its constraints into CSP represented as CPLEX java object. The solve() call is executed, after which the native libraries are invoked and solve the task. The results are printed on the screen. The whole process is shown in Fig. 3.3.

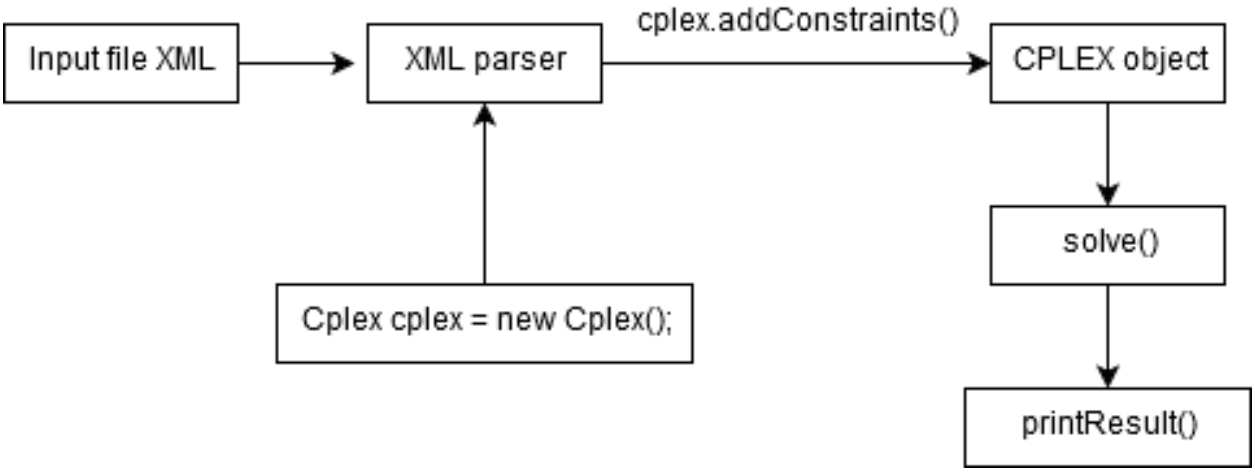



FIGURE 3.3: Program functionality

## 3.5 Deployment

ALL IBM CPLEX optimization algorithms are implemented in native libraries. Native libraries are such libraries that are executed in the central processing unit (CPU) and are platform dependent components of the solution. Therefore, for system installation it is needed to download and install the latest version of CPLEX tool for the target performance. Since CPLEX tool is being developed and maintained independently from my solution, one of the design goals is to ensure possibility to upgrade CPLEX and substitute it with a newer version. For this reason .jar supplied by IBM has to be decoupled from my solution. If my solution would include CPLEX.jar as a dependency, upgrading CPLEX would be problematic. To ensure upgrades my solution .jar and cplex.jar should be located in the same classpath. Besides the Java Virtual Machine (JVM) needs to know where to find the system libraries. For this reason JVM is configurable with –Djava.library.path parameter, which should point inside cplex installation directory(platform dependent) containing shared system libraries of CPLEX.

```
java -cp "..." -Djava.library.path="..." package.MainClass
```

Listing 3.2: Example of command to run program in Windows

```
java -cp "..." -Djava.library.path="..." package.MainClass
```

Listing 3.3: Example of command to run program in Unix

The classpath in Listings 3.2 and 3.3 must contain absolute system paths to cplex.jar archive inside IBM CPLEX installation directory and to .jar archive with the program.
-Djava.library.path parameter of the JVM should be set to the absolute system path to the native libraries inside IBM CPLEX installation directory.

# Open Questions and Future Work

Obtaining the minimal cost is a constrained nonlinear minimization problem. The objective function is nonlinear, nonconvex, not twice continuously differentiable, but has subgradients. This thesis investigated possibilities to solve this problem by means of MILP and CSP. The MILP approach didn't work out. The CSP approach produces results, however the decorated values depend on the attack tree representation – is not representation invariant. Representation invariance is one of the properties required by Mauw–Oostdijk attack tree foundations [8]. They require that computational methods produce identical results for semantically equivalent inputs. Determining attack feasibility doesn't violate this requirement, however the decoration does. At the moment it is not known what are the effects of violating representation invariance during attack tree decoration. It seems that it doesn't affect the results of the feasibility analysis. The effects of this violation is an open question and need to be studied further.

The tool prototype was created. It takes an input XML file formatted in ADTool notation, converts it to CSP and launches IBM CPLEX solver to obtain results. Different optimization tools were considered and IBM CPLEX was selected for the purposes of the thesis. The prototype produced in this thesis seamlessly integrates the CPLEX optimizer and GUI written in JavaFX into a solid proof of concept prototype demonstrating possibility to execute attack tree feasibility analysis combining available statistical data as well as expert estimations.

In case we need to preserve representational invariance in attack tree decoration we have several options, although they come at a cost of increased computational complexity and/or precision. Due to the fact that the objective function is not twice continuously differentiable, but contains some gradients such a nonlinear objective function may be minimized using primal–dual interior–point method for nonlinear optimization. Interior–point methods, also known as barrier methods solve nonlinear convex optimization problems by encoding a feasible set using a barrier function (typically logarithmic barrier function). To the best of the author's knowledge there exist no Java libraries implementing interior–point method. Interior–point solvers are known to exist in WolframAlpha and MATLAB.

Another alternative would be to use subgradient method – an algorithm for minimiz-

ing a non–differentiable convex function. Attempts to find a suitable solver implementing this algorithm were not successful.

Yet another alternative would be to apply sequential quadratic programming (SQP) and sequential linear programming (SLP) – iterative methods for nonlinear optimization which solve optimization problems by solving a simplified problem during each step (quadratic or linear), and incorporating the optimal results of each step into the overall optimization problem. These methods are known to work for the cases when the objective function as well as constraints are twice continuously differentiable – which is not the considered problem. It is not known whether these methods will work with functions which are not twice continuously differentiable, but have subgradients.

# Appendix A

# Attack–Defense Tree XSD Schema

```
1  <?xml version="1.0"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:element name="adtree" type="adtreeType">
4    </xsd:element>
5    <xsd:complexType name="adtreeType">
6      <xsd:sequence>
7        <xsd:element name="node" type="nodeType" minOccurs="1" maxOccurs="1"/>
8      </xsd:sequence>
9    </xsd:complexType>
10    <xsd:complexType name="nodeType">
11      <xsd:sequence>
12        <xsd:element name="label" type="labelType" minOccurs="1" maxOccurs="1"/>
13        <xsd:element name="node" type="nodeType" minOccurs="0" maxOccurs="unbounded"/>
14      </xsd:sequence>
15      <xsd:attribute name="refinement" type="refinementType" use="optional" default="disjunctive"/>
16      <xsd:attribute name="switchRole" type="booleanType" use="optional" default="no"/>
17    </xsd:complexType>
18    <xsd:simpleType name="labelType">
19      <xsd:restriction base="xsd:string">
20        <xsd:pattern value="[0-9A-Za-z\s\?!\-_]+"/>
21        <xsd:whiteSpace value="preserve"/>
22      </xsd:restriction>
23    </xsd:simpleType>
24    <xsd:simpleType name="refinementType">
25      <xsd:restriction base="xsd:string">
26        <xsd:enumeration value="disjunctive"/>
27        <xsd:enumeration value="conjunctive"/>
28      </xsd:restriction>
29    </xsd:simpleType>
30    <xsd:simpleType name="booleanType">
```

```
31      <xsd:restriction base="xsd:string">
32        <xsd:enumeration value="yes"/>
33        <xsd:enumeration value="no"/>
34      </xsd:restriction>
35    </xsd:simpleType>
36  </xsd:schema>
```

Listing A.1: Schema adtree.xsd for XML files without attributes [6]

# Appendix B

# Extended Attack–Defense Tree XSD Schema

---

```
1  <?xml version="1.0"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:element name="adtree" type="adtreeType">
4    </xsd:element>
5    <xsd:complexType name="adtreeType">
6      <xsd:sequence>
7        <xsd:element name="node" type="nodeType" minOccurs="1" maxOccurs="1"/>
8      </xsd:sequence>
9    </xsd:complexType>
10    <xsd:complexType name="nodeType">
11      <xsd:sequence>
12        <xsd:element name="label" type="labelType" minOccurs="1" maxOccurs="1"/>
13        <xsd:element name="node" type="nodeType" minOccurs="0" maxOccurs="unbounded"/>
14      </xsd:sequence>
15      <xsd:attribute name="refinement" type="refinementType" use="optional" default="disjunctive"/>
16      <xsd:attribute name="switchRole" type="booleanType" use="optional" default="no"/>
17    </xsd:complexType>
18    <xsd:simpleType name="labelType">
19      <xsd:restriction base="xsd:string">
20        <xsd:pattern value="[0-9A-Za-z\s\?!\-_]+"/>
21        <xsd:whiteSpace value="preserve"/>
22      </xsd:restriction>
23    </xsd:simpleType>
24    <xsd:simpleType name="refinementType">
25      <xsd:restriction base="xsd:string">
26        <xsd:enumeration value="disjunctive"/>
27        <xsd:enumeration value="conjunctive"/>
```

```
28      </xsd:restriction>
29    </xsd:simpleType>
30    <xsd:simpleType name="booleanType">
31      <xsd:restriction base="xsd:string">
32        <xsd:enumeration value="yes"/>
33        <xsd:enumeration value="no"/>
34      </xsd:restriction>
35    </xsd:simpleType>
36   <xsd:simpleType name="constraintType">
37     <xsd:restriction base="xsd:string">
38       <xsd:pattern value="(EQ|GE|LE)\s+[0-9]+([.]{1}[0-9]+)*"/>
39       <xsd:whiteSpace value="preserve"/>
40     </xsd:restriction>
41   </xsd:simpleType>
42 </xsd:schema>
```

Listing B.1: Improved Schema adtree.xsd for XML files without attributes [6]

# References


[1] Bagnato, A., Kordy, B., Meland, P.H., Schweitzer, P.: Attribute decoration of attack-defense trees. IJSSE **3**(2) (2012) 1–35

[2] Böhme, R.: Security metrics and security investment models. In Echizen, I., Kunihiro, N., Sasaki, R., eds.: Advances in Information and Computer Security - 5th International Workshop on Security, IWSEC 2010, Kobe, Japan, November 22-24, 2010. Proceedings. Volume 6434 of Lecture Notes in Computer Science., Springer (2010) 10–24

[3] Dimkov, T., Pieters, W., Hartel, P.H.: Laptop theft: a case study on the effectiveness of security mechanisms in open organizations. In Al-Shaer, E., Keromytis, A.D., Shmatikov, V., eds.: Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010, ACM (2010) 666–668

[4] Jaquith, A.: Security Metrics: Replacing Fear, Uncertainty, and Doubt. 1st edn. Addison-Wesley Professional, Upper Saddle River, NJ (4 2007)

[5] Kordy, B., Piètre-Cambacédès, L., Schweitzer, P.: Dag-based attack and defense modeling: Don't miss the forest for the attack trees. Computer Science Review **13-14** (2014) 1–38

[6] Kordy, P., Schweitzer, P.: The ADTool Manual. (2015) `http://satoss.uni.lu/members/piotr/adtool/manual.pdf\ newline(Accessedon11.05.2017).`

[7] Lenin, A.: Reliable and Efficient Determination of the Likelihood of Rational Attacks. TUT Press, Tallinn (2015)

[8] Mauw, S., Oostdijk, M.: Foundations of attack trees. In: International Conference on Information Security and Cryptology ICISC 2005. LNCS 3935, Springer (2005) 186–198

[9] Meyers, C.A., Powers, S.S., Faissol, D.M.: Taxonomies of cyber adversaries and attacks: A survey of incidents and approaches. Technical Report LLNL-TR-419041 TRN:

US200924%%162, Lawrence Livermore National Laboratory (LLNL), Livermore, CA (10 2009)

[10] Raymond, J.: Traffic analysis: Protocols, attacks, design issues, and open problems. In Federrath, H., ed.: Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000, Proceedings. Volume 2009 of Lecture Notes in Computer Science., Springer (2000) 10–29

[11] Salles-Loustau, G., Berthier, R., Collange, E., Sobesto, B., Cukier, M.: Characterizing attackers and attacks: An empirical study. In Alkalai, L., Tsai, T., Yoneda, T., eds.: 17th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2011, Pasadena, CA, USA, December 12-14, 2011, IEEE Computer Society (2011) 174–183

[12] Schneier, B.: Attack trees. Dr. Dobb's Journal (12 1999)

[13] Schneier, B.: Secrets & Lies. Dpunkt.Verlag GmbH. Wiley-VCH, Heidelberg Weinheim (8 2004)

[14] Vose, D.: Risk Analysis: A Quantitative Guide. 3rd edn. Wiley, Chichester, England Hoboken, NJ (4 2008)

[15] Weiss, J.D.: A system security engineering process. In 14th Nat. Comp. Sec. Conf. (1991) 572–581.

[16] Council of the European Union: Council regulation (EU) no 40/2013 (2013) http://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX%3A32013L0040 (Accessed on 05.02.2017).

[17] Glpk (gnu linear programming kit) https://www.gnu.org/software/glpk/ (Accessed on 05.04.2017).

[18] Glpk for java http://http://glpk-java.sourceforge.net/ (Accessed on 05.04.2017).

[19] Gurobi optimizer http://www.gurobi.com/products/gurobi-optimizer (Accessed on 05.04.2017).

[20] Ibm ilog cplex optimization studio community edition https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/ (Accessed on 05.05.2017).

[21] Publications, I.I.B., ed.: European Union Cyber Security Strategy and Programs Handbook: Strategic Information and Regulations. Updated edn. Intl Business Pubns USA (7 2013)