predict
prioritise
prevent

TRESPASS

Technology-supported Risk Estimation by Predictive Assessment of Socio-technical Security

Deliverable D3.3.2

Methods for Stochastic Analysis

Project:	TRE$_S$PASS
Project Number:	ICT-318003
Deliverable:	D3.3.2
Title:	Methods for Stochastic Analysis
Version:	1.0
Confidentiality:	Public
Editor:	Z. Aslanyan
Cont. Authors:	R.Kumar, A.Lenin, M.Martins, R.R.Hansen
Date:	2015-10-30

Part of the Seventh Framework Programme
Funded by the EC-DG CONNECT

Members of the TRESPASS Consortium

1.	University of Twente	UT	The Netherlands
2.	Technical University of Denmark	DTU	Denmark
3.	Cybernetica	CYB	Estonia
4.	GMV Portugal	GMVP	Portugal
5.	GMV Spain	GMVS	Spain
6.	Royal Holloway University of London	RHUL	United Kingdom
7.	itrust consulting	ITR	Luxembourg
8.	Goethe University Frankfurt	GUF	Germany
9.	IBM Research	IBM	Switzerland
10.	Delft University of Technology	TUD	The Netherlands
11.	Hamburg University of Technology	TUHH	Germany
12.	University of Luxembourg	UL	Luxembourg
13.	Aalborg University	AAU	Denmark
14.	Consult Hyperion	CHYP	United Kingdom
15.	BizzDesign	BD	The Netherlands
16.	Deloitte	DELO	The Netherlands
17.	Lust	LUST	The Netherlands

Document History

Authors		
Partner	Name	Chapters
DTU	Zaruhi Aslanyan	All
ITR	Miguel Martins	2.3, 4.6, 4.7, 4.8
UT	Rajesh Kumar	2.2, 4.1, 4.2, 4.8
CYB	Aleksandr Lenin	4.5, 4.8
AAU	René Rydhof Hansen	4.4

Quality assurance		
Role	Name	Date
Editor	Zaruhi Aslanyan	2015-10-30
Reviewer	Wolter Pieters	2015-10-16
Reviewer	Sören Bleikertz	2015-10-16
Reviewer	Dieter Gollmann	2015-10-16
Task leader	Christian W. Probst	2015-10-30
WP leader	Jaco van de Pol	2015-10-30
Coordinator	Pieter Hartel	2015-10-30

Circulation	
Recipient	Date of submission
Project Partners	2015-10-30
European Commission	2015-10-30

Acknowledgement: *The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318003 (TRE$_S$PASS). This publication reflects only the authors' views and the Union is not liable for any use that may be made of the information contained herein.*

Contents

List of Figures

List of Tables

Management Summary

Key takeaways: This document is the final deliverable of Task T3.3. The key takeaways from this document are:

- We have continued the work on the methods presented at M12 in (The TRESPASS Project, D3.3.1, 2013). Significant progress has been made on efficient quantitative analysis methods for attack trees, both by improving some of the presented techniques and by developing new ones;
- Novel techniques have been successfully developed, notably based on priced time automata, on Pareto optimality and on statistical model checking;
- The presented techniques can answer a plethora of different questions such as "What are the most probable attacks", "What is the maximal damage an attacker can do", "What are the attacks with the maximum probability and the minimum cost", and "What is the minimal time needed to complete a successful attack within a given budget".
- We have continued to apply existing methods to the problems faced in the TRESPASS project, evaluating the developed techniques on problems from the case studies.

The *Quantitative analysis tools* work package within the TRESPASS project aims to devise algorithms and tools to analyse socio-technical security models with respect to quantitative security measures. Socio-technical models will be provided by WP1, whose low level semantic models will be identified as appropriate quantitative models. In particular, specific properties will be of interest in the systems under investigation, for instance "What are the most probable attacks?", "What is the minimal time needed to complete a successful attack within a given budget?" Given the quantitative model and the property, formal analysis techniques such as model checking or static analysis will be applied to the quantitative model.

Due to their wide-spread use in security engineering, WP3 uses attack trees as the modelling formalism to reason about the dependency and possibility of different attack vectors. Moreover, we consider a wide range of quantitative security properties, such as (1) probability, both discrete and continuous; (2) time; and (3) cost/reward-based quantities (effort, cost, impact). On the one hand, these classes emerge as key aspects in many socio-technical security problems; on the other hand, powerful tools exist that are able to analyse them.

This deliverable presents the progress in the area of stochastic analysis methods in months 13-36 of the TRESPASS project. It should be read in close conjunction with (The TRESPASS Project, D3.2.1, 2015), where D3.2.1 gives a detailed description of the extraction of stochastic models analysed in this deliverable.

We first present an overview of techniques for quantitative analysis of attack scenarios. While a large amount of methods has been discussed in the previous deliverable (The TRESPASS Project, D3.3.1, 2013), here we focus on new and improved analysis techniques, in particular

- Statistical model checking for analysis of complex models by simulation.
- Pareto efficiency for optimising attack trees with multiple attributes.
- UPPAAL model checker for computing optimal traces.

For each of these novel techniques, we give a brief overview of its key features and underlying idea and how it can be used to answer crucial questions in the security context. We complete the presentation with a comparison of the featured techniques with respect to their input, their output and strengths.

1. Introduction

1.1. Motivation and Challengies

The goal of WP3, quantitative analysis tools, is to devise algorithms that are able to compute the security properties of the socio-technical security models developed in WP1, using stochastic model checking (SMC) and static analysis. This allows us to evaluate and compare certain attack scenarios and countermeasures with respect to quantitative security properties. Typical questions that can be answered are: “What are the most probable attacks?”, “What is the maximal damage an attacker can do?”, “What are the attacks with the maximum probability and the minimum cost?”, and “What is the minimal time needed to complete a successful attack within a given budget?” On the one hand, these questions consider the key aspects in many socio-technical security problems; on the other hand, they allow to rank and identify the most occurring attacks based on various quantitative security properties.

There is a tight connection between WP3 and the other work packages. The analysis techniques developed in WP3 are used to evaluate the socio-technical security model developed in WP1 (Socio-technical security model specification). The model enriched with data collected by WP2 (Data management process), and the analysis results are then visualised by WP4 (visualisation process and tools). The quality and applicability of the developed techniques are validated by case studies provided in WP7. Furthermore, the set of algorithms and analysis techniques is integrated into one tool, which is then further integrated into a larger framework provided by WP6.

Internally, WP3 is divided into five tasks: Task T3.1 identifies and collect requirements. Task 3.2 extracts a formal attack model from the socio-technical model from WP1 or from attack tree-based models that are manageable to formal analysis, such as model checking and static analysis. Task T3.3 analyses the quantitative low-level model with respect to security properties using existing and newly developed analysis methods, and Task T3.4 will interpret the results in the socio-technical security model. Finally, Task T3.5 deals with dynamic properties of models.

Both model checking and static analysis have been used with great success for verification and validation purposes, and here we will adapt and extend these methods to also generate traces of actions that will potentially (or probably) lead to a breach of the security goals stated in the socio-technical security model. For any non-trivial policy model there will likely be a large number of potential attack scenarios. An important task of this work package is to develop ways to systematically and robustly reduce the number of spurious and irrelevant attack scenarios, e.g., small variations over the same basic attack, as well

as methods for ranking the generated attacks, e.g., according to likelihood of success of attack.

Attack trees are a suitable tool for presenting socio-technical threats and conveying security information to non-experts. Most of the analysis techniques developed in WP3 uses attack tree-based formalisms for evaluating the security properties of the model. Even though attack trees are a widely used and intuitive tool, there is not one perfect way for representing attack scenarios. That is why together with analysis techniques using attack tree-based formalisms, we present also analysis techniques that use different models such as timed automata. Extraction of such models and their formal semantics, either directly form the socio-technical model or from an attack tree-based model, is presented in more details in (The TRE$_S$PASS Project, D3.2.1, 2015).

This deliverable is the final report on work performed in Task T3.3. It extends the work presented in (The TRE$_S$PASS Project, D3.3.1, 2013) and illustrates the novel and improved analysis techniques for providing qualified assessment and ranking of attacks based on the security properties.

The tools for some analysis techniques are visually presented in videos available in `https://vimeo.com/channels/trespass` (The TREsPASS Project, 2015). Finally, we refer the reader to D6.2.2, final refinement of TRE$_S$PASS functional requirements (The TRE$_S$PASS Project, D6.2.2, 2015), as useful prior reading material.

Focus

As elaborated in (The TRE$_S$PASS Project, D3.1.1, 2013), WP3 focuses on three main classes of quantities

1. probability,
2. time and
3. effort/cost.

These quantities emerge as key aspects in many socio-technical security problems under consideration in WP1. Moreover, we identify various analysis techniques in the tools that are able to handle these quantities.

Probability versus Likelihood

Within the TRE$_S$PASS project, we use the word likelihood to refer to a quantitative estimation for the chance of a certain event to happen: How likely is a certain attack to happen? How likely is it that it succeeds? In WP3, we use the word probability to refer to the standard mathematical concept, formalised in probability theory. We can translate informal likelihoods in several ways: a given likelihood can, depending on the context and purpose, be formalised either as a single probability value; as a range of probabilities (“between 0.10 and 0.11”), or as a continuous probability distribution (e.g. a Gaussian distribution

whose mean is given by the likelihood). A proper formalisation is part of the challenges of WP3 and WP1.

1.2. Goals

The goal of this deliverable is to document the final TRE$_S$PASS methods for stochastic analysis, consolidating the findings of Task T3.3.

This deliverable focuses on the novel and improved analysis techniques with respect to the ones discussed in (The TRE$_S$PASS Project, D3.3.1, 2013). Most of the techniques presented in (The TRE$_S$PASS Project, D3.3.1, 2013) consider attack tree-based models, while in this deliverable novel approaches based on time automata and Markov chains are presented. Moreover, as an attacker property might have effect on various quantities, e.g. probability of success or required time to complete an attack, the concept of attacker profile has been considered in new analysis techniques.

In particular, we focus on the following:

- Present an overview of analysis methods that predated the TRE$_S$PASS project.
- Discuss the novel and improved techniques developed during the months 13-36 in the TRE$_S$PASS project.
- Compare these techniques with respect to their capabilities, formal techniques used, and whether a tool implementation is available.

1.3. Foreground and background

The background the work performed within WP3 builds upon is presented in Chapters 2 and 3. We provide a brief summary of the most important concepts related to tree-based and other extracted models in Chapter 2. In Chapter 3, we summarise and classify existing techniques for quantitative analysis of models. Most of the material in Chapter 2 is new (around 75%).

Chapter 4 presents the foreground obtained during the months 13-36 of the project. In Chapter 4, the core of the document, we report on the methodologies that the WP3 team developed. For each of these novel techniques, we present an underlying motivation, describe its key features and main advantages.

1.4. Document structure

We start with a short description of the considered models and existing methods in Chapters 2 and 3. In Chapter 4 we present and discuss various analysis methods developed during the TRESPASS project. We compare these techniques with respect to their capabilities, formal techniques used, complexity, and whether a tool implementation is available. Chapter 5 concludes the document.

Appendix A provides the context for this deliverable in the TRESPASS project. It describes the overall summary of the project and the TRESPASS workflow.

2. State of the Art

In this chapter, we briefly present the security models that have been used by WP3. We motivate our selection and provide pointers to relevant literature references.

2.1. Attack Tree-based Formalisms

2.1.1. Attack Trees

In 1999, Schneier popularised attack trees as a graphical modelling tool to evaluate the security of complex systems (Schneier, 1999). The root of an attack tree corresponds to an attacker's goal. This goal is refined using a tree structure into sub-goals. The refinement of a node is either disjunctive or conjunctive. The goal of a disjunctively refined node, also called OR node, is achieved when at least one of its children's goals is achieved. The goal of a conjunctively refined node, also called AND node, is achieved when all of its children's goals are achieved. The leaves of the tree represent the actions to be executed by the attacker. They are often called basic or atomic actions.

Attack trees are currently one of the most popular graphical modelling techniques and are widely accepted and used by industry. Notable applications include the security analysis of: supervisory control and data acquisition (SCADA) systems (Byres, Franz, & Miller, 2004; Ten, Liu, & Manimaran, 2007; Tanu & Arreymbi, 2010), voting systems (Lazarus, Dill, Epstein, & Hall, 2011; Buldas & Mägi, 2007), vehicular communication systems (Henniger et al., 2009; Aijaz et al., 2006), Internet related attacks (Tidwell, Larson, Fitch, & Hale, 2001; Lin, Zavarsky, Ruhl, & Lindskog, 2009), secure software engineering (Jung, Elberzhager, Bagnato, & Raiteri, 2010), and socio-technical attacks (Bagnato, Kordy, Meland, & Schweitzer, 2012; Eom, Park, Park, & Chung, 2011; Reddy, Venter, Olivier, & Currie, 2008).

2.1.2. Attack–Defence Trees

An obvious limitation of attack trees is that they cannot capture the interaction between attacks carried out on a system and the defences that could be put in place to fend off the attacks. This consequently limits the precision with which the best defensive strategies can be analysed, since it does not take into account the effects of potential defensive measures which would need to be overcome by new attacks. Similarly, a regular attack tree does not allow for the visualisation and consideration of the evolution of a system's

security, because the evolution can only be understood in view of both the attacker's as well as the defender's actions.

In order to overcome these limitations, attack–defence trees (ADTrees) have been introduced by Kordy et al. in (Kordy, Mauw, Radomirović, & Schweitzer, 2011). In brief, an attack–defence tree (ADTree) is a node-labelled rooted tree describing the measures an attacker might take in order to attack a system and the defences that a defender can employ to protect the system. Formally, ADTrees have nodes of two opposite types: attack nodes and defence nodes, which correspond to an attacker's and a defender's (sub-)goals, respectively. As in attack trees, a node of an ADTree can be conjunctively and disjunctively refined using child nodes of the same type. Additionally, each node may also have a child of the opposite type, representing a countermeasure. Thus, an attack node may have several children which refine the attack and one child which defends against the attack. The defending child, in turn, may have several children which refine the defence and one child that is an attack node and counters the defence (Kordy, Mauw, Radomirović, & Schweitzer, 2012).

2.2. State space based formalisms

State space based formalisms are basically transition systems (Continuous time Markov chain/Interactive Markov chain /Markov automata /Timed automata), used to capture the dynamics of system behaviour. They are quite flexible and expressive in nature as they are not tied to particular syntactic domain-specific constructs and thus enable us to model the system's temporal, stochastic dependencies and non-deterministic behaviour (Boudali, Haverkort, Kuntz, & Stoelinga, 2007; Hartmanns & Hermanns, 2015).

To combine the intuitive representation of attack trees with the modelling power of transition systems, we extract a small stochastic model for each element in an attack tree (The TRE$_S$PASS Project, D3.2.1, 2015), that is further analysed as part of this task. The choice of extraction method is tightly coupled to the quantitative metric of interest. While we can reason about time dependent attacks by deriving I/O interactive Markov chains and Markov automata from attack trees, we provide optimum attack values and paths by transforming attack trees into weighted timed automata.

2.2.1. I/O Interactive Markov chain

Interactive Markov chains (Hermanns, 2002) combine the stochastic behaviour with non deterministic choices but lack probabilistic choices. I/O interactive Markov chains extend interactive Markov chain by having input and output action labels (Lynch & Tuttle, 1988), over which the transition systems can synchronize with each other. Thus, they are compositional in nature and are a quite expressive formalism. Here, they can be used to model the attacker behaviour of eventually obtaining success after a given probabilistic distribution of time (CDF). They have basically two transitions:

- Markovian transition: models the stochastic behaviour, given by the rate of the probability distribution and is labelled with λ.
- Action transition: is either labelled with action labels $Activated$ or an internal action τ. Input Actions labelled with $Activated?$ can synchronize with output actions labelled with action labels $Activated!$ and are executed in zero time (as in the CSP approach in process algebra) (Hoare, 1985).

When both Markovian and Actions transition are enabled, it is governed by maximal progress – i.e. internal transitions cannot be delayed over Markovian transitions, though external transitions can be infinitely delayed.

The analysis methods using I/O interactive Markov chains has been already discussed and the results have been validated through the IPTV case study in (The TRESPASS Project, D3.3.1, 2013). Hence, this method is not detailed in this deliverable. However, a brief overview of the extraction methodology of I/O IMC from attack trees has been provided in (The TRESPASS Project, D3.2.1, 2015).

2.2.2. I/O Markov automata

Input/output Markov automata (I/O-MAs) extend Markov automata (MAs) (Guck, Hatefi, Hermanns, Katoen, & Timmer, 2013) by integrating the action behaviour of input/output automata (Lynch & Tuttle, 1988). We classify transitions as either *Markovian* or *probabilistic*. A Markovian transition is labelled with a rate λ, indicating that this transition can be taken after an exponentially distributed delay with parameter λ. Thus, the probability to take the transition within time t equals $1 - e^{-\lambda t}$.

A probabilistic transition leads to a probability distribution μ, that is, we move to the next state s with probability $\mu(s)$. Probabilistic transitions are labelled with different kinds of actions: (1) *Input actions* (denoted a?) can only be taken if another I/O-MA executes an output action a!; we say that a? requires *synchronisation* on a!. The action is thus controlled by the environment which can delay it; (2) *Output actions* (denoted a!) cannot be delayed and are controlled by the respective I/O-MA. Transitions labelled with output actions have to be taken immediately; (3) *Internal actions* (denoted a;) are controlled by the respective I/O-MA and cannot be delayed, quite like output actions. However, they are not visible to the environment. I/O-MAs are *input-enabled*, meaning that all states of a particular I/O-MA can respond to any of its inputs by means of an outgoing transition labelled with the respective input action.

The beauty of this formalism lies in the possibility to construct large I/O-MAs from several interacting smaller ones by composing them in parallel. We denote with $M_1||M_2$ the parallel composition of I/O-MAs M_1 and M_2. Then $M_c = M_1||M_2$ is again an I/O-MA (with the Cartesian product of M_1 and M_2 as state space) expressing the joint behaviour of its constituents: (1) If an action does not require synchronisation, then M_1 and M_2 evolve independently; (2) If an action a on an probabilistic transition requires synchronisation, then both I/O-MAs must be able to perform the output action a! and corresponding input action a? at the same time.

2.2.3. Priced timed Automata

Timed automata (Alur & Dill, 1994a), are labelled transition systems with real valued integers called *clocks* that synchronously increase over time. These clocks serve as *guards* over transitions serving as time bound after which an edge has to be necessarily taken or as invariants in locations, which implies that the location needs to be left after a residence time. Since they are extensions of labelled transition systems (LTS), they are compositional in nature.

Priced/Weighted timed automata (PTA) (Behrmann, Larsen, & Rasmussen, 2005a) extend timed automata, by adding costs to locations and actions transitions. Costs can either be accumulated in states, proportionally to the residence time, or by taking a transition. Both, TA and PTA can be analysed for a wide number of properties in model checking, including absence of deadlocks, safety, and liveness.

2.3. Asset-based Formalisms

2.3.1. TRICK Service

TRICK Service is a risk analysis and treatment tool which is used in the context of Information security risk management. TRICK is acronym for “Tool for Risk management of an ISMS based on Central Knowledge base”.

TRICK Service allows the determination, for an information system and its associated context, a list of security measures to be put in place in order to reduce losses caused by the occurrence of threats.

The methodology is based on ISO/IEC 27005 and is based on the following major steps:

- Context definition
- Asset identification
- Risk scenario identification
- Risk estimation
- Inventory of the measures
- Risk specificity

A detailed description of the tool and related methodology can be found in (The TRE$_S$-PASS Project, D5.2.1, 2014).

3. Background on Quantitative Analysis of Security

In this chapter, we briefly present some of the techniques for quantitative analysis of security scenarios. A more detailed explanation is given in (The TRESPASS Project, D3.3.1, 2013).

Various models, such as attack tree-based models, not only allow for a systematic modelling of security scenarios. They also assist in quantitative analysis of such scenarios and support the risk estimation process. Existing techniques for quantitative analysis of security scenarios can be divided into two main classes:

Time-abstract analysis Analyses an attack by considering it as a static system which does not temporally evolve. Typically questions that are answered are "Which is the best attack vector for the attacker given his preferences" and "Which is the cheapest attack that succeeds in more than 10% of all cases".

Time-dynamic analysis Analyses an attack by taking its evolution over time into account. The execution of attack steps consumes time, and the order of execution matters due to time dependencies such as sequencing. With this technique one can investigate "What is the probability to penetrate the system in less than 1 day".

Time-abstract analysis can be further classified into

1. Bottom-up analysis techniques (for attack trees)
2. Game-based analysis techniques, briefly discussed in Section 3.3.

The bottom-up analysis of attack trees exists in two flavours:

- Standard bottom-up single-parameter attack tree analysis, briefly described in Section 3.1, and
- Multi-parameter analysis of attack trees, summarised in Section 3.2.

3.1. Singe-Parameter Bottom-up Analysis

The single-parameter attack tree analysis was first intuitively sketched by Schneier in (Schneier, 1999) and then subsequently formalised by Mauw and Oostdijk in (Mauw & Oostdijk, 2006). When using this approach, a user decorates each leaf of the attack tree with one parameter, e.g., cost, and a bottom-up algorithm propagates the parameter toward the root, using appropriate mathematical operators. The standard single-parameter bottom-up evaluation method has been extended from attack trees to ADTrees in (Kordy, Mauw, Radomirović, & Schweitzer, 2012; Kordy, Mauw, & Schweitzer, 2012). Detailed description and a general taxonomy of the measures supported by the bottom-up approach can be found in (Kordy, Mauw, & Schweitzer, 2012).

3.2. Multi-Parameter Bottom-up Analysis

Although the single-parameter bottom-up procedure is commonly used to perform computations with attack trees, see (Bagnato et al., 2012) and (Kordy, Piètre-Cambacédès, & Schweitzer, 2013) for an overview, it reasons about individual parameters, such as time or cost of an attack, in separation from other quantities. In practice, however, different parameters influence each other. For instance, the time and the probability attributes are closely related and should not be analysed in isolation. The same applies to the cost, gain and penalty parameters. This is why the multi-parameter analysis techniques have been developed for attack trees.

In the multi-parameter approach (Buldas, Laud, Priisalu, Saarepera, & Willemson, 2006a; Jürgenson & Willemson, 2008; Willemson & Jürgenson, 2010; Buldas & Stepanenko, 2012a; Buldas & Lenin, 2013), every leaf of an attack tree may be assigned an arbitrary number of parameters. This approach does not propagate leaves' parameters directly – instead, a value called utility, not directly related to any of the elementary attack parameters, is calculated for each leaf. Once done, attacker utility is propagated towards the root node of the attack tree, similarly to the standard bottom-up approach. Utility is the result of a function of the parameter estimations of the atomic attacks. The logic behind the function is model-specific and varies from model to model.

3.3. Game-based Analysis

An alternative to the bottom-up approaches is the game-theoretic analysis of attack trees, which does not rely on the actual structure of the attack tree, but instead considers the Boolean function of it (Buldas et al., 2006a; Jürgenson & Willemson, 2008; Willemson & Jürgenson, 2010; Buldas & Stepanenko, 2012a). If an atomic attack succeeds, the corresponding variable in the Boolean function has the value 1 (true), and false otherwise. The analysis adds the satisfiability requirement – the sets of atomic attacks (variables of the Boolean function) are considered solutions, if being assigned value 1 (true), satisfy the

Boolean function of an attack tree. If an attacker will launch attacks from such a satisfying set and they succeed, the attacker will reach his goal, similarly to reaching the root node of an attack tree.

3.4. Summary

The survey paper (Kordy et al., 2013) contains a systematic review of existing techniques for quantitative analysis of attack tree-based formalisms. Analysis performed in (Kordy et al., 2013) shows that most of the existing quantitative frameworks are not able to deal properly with models containing dependent actions. Furthermore, many analysis techniques have been designed to deal with very specific application domains, e.g., intrusion detection or secure software development, and they are inappropriate for performing computations relevant for the TRE$_S$PASS considerations. Finally, most of the analysis are done on attack tree-based formalisms and only a small amount of work evaluates security scenarios through state space based formalisms. These conclusions motivated us to propose novel computational frameworks for analysing security scenarios, which we describe in the next chapter.

4. TRESPASS Analysis Techniques

In this chapter, we present the new and improved techniques for quantitative analysis of security models that the WP3 team developed. For each of the presented techniques:

- We first motivate why it is useful and how it improves upon the existing techniques.
- Then, we briefly present the developed technique and associated models.
- We continue with discussing the underlying formal methodology.
- We demonstrate the presented techniques on a case study from the TRESPASS project or the existing literature.
- We conclude by summarising the key features of the technique.

For most of the presented analysis techniques a supported tool has been developed by the TRESPASS partners. Some of the tools are illustrated and discussed in details through the videos accessible in https://vimeo.com/channels/trespass (The TREsPASS Project, 2015).

4.1. Quantitative Attack Tree Analysis via Priced Timed Automata

4.1.1. Motivation

We use the priced timed automata formalism (refer section 2.2.3) to analyse ATs and answer questions such as : (1) Given an adversary persona (refer section 4.1.2), what is the minimal time / resources, or skill level needed to complete a successful attack? (2) Which attack path (a subtree of attack tree) will a rational attacker choose, if he wants to reach his goal minimising his incurred cost?

The answers to these questions with precise values are based on developing an optimisation framework which can take care of multiple attributes in an attack tree which are causally and temporally dependent. Frameworks such as (Buckshaw, 2014; LeMay, Ford, Keefe, Sanders, & Muehrcke, 2011) do exist for security modelling but are based on simulating attacker behaviour to obtain an estimate value. While their focus is to prevent an attack, our methodology is geared towards preventing as well as prioritising the attack scenarios.

Thus, our analysis framework can provide several useful metrics such as :

(1) *Attack values.* Given an attack tree, and a quantity of interest, we can compute its value: What is the minimal time, resources, or skill level needed to complete a successful attack? What is the maximal damage an attacker can do? (2) *Attack paths.* Apart from the value of an attack, it is very useful to know the attack path leading to the optimal attack. Note that the path is in fact a subtree, since optimal attacks often carry out several steps in parallel. (3) *Ranking.* Apart from computing the optimal attack values and path, we can also determine the top-10 of worst attacks, which is very important to determine appropriate countermeasures. (4) *Pareto-optimal curves* that show trade-offs when multiple objectives conflict. For instance, what is the minimal time needed to complete a successful attack within a given budget? What is the maximal damage that can be incurred in one year?

4.1.2. Methodology and model

The input to our analysis method is an annotated attack tree (with attributes such as cost and time to successfully execute the atomic attack steps) and an adversary who possesses specific skills, budget and knowledge in performing these attack steps in reaching his goal. Technically, our framework is realised via UPPAAL CORA (Behrmann, Larsen, & Rasmussen, 2005b) : we translate each attack tree gate and leaf into a priced timed automata (PTA). Together these form a network of PTAs representing the entire attack tree. This modular approach yields a flexible framework that can easily be extended with future needs, such as countermeasures.

Attacker profile. An attacker profile is a persona/institution, who is motivated to reach the asset (attack goal in attack tree) by executing basic attack steps in his intended order of preference. He starts with a limited budget, uses his skills and invests resources (time, cost) in executing basic attack steps and eventually fails or succeeds with a certain probability. Personas offer a means of illustrating the different stakeholders and perspectives in a risk scenario (The TRE$_S$PASS Project, D2.3.2, 2015; Buckshaw, 2014; LeMay et al., 2011). It includes type of attacker (malicious insider/ disgruntled employee/ competitor/ irrational individual), his capabilities, resources, initial knowledge of system and having an initial access to the system. Currently, we use pre-defined persona and their utilities based on extensive brainstorming for our case studies to demonstrate our proof of concept.

4.1.3. Example

To establish our methodology, we take a small attack tree as in Figure 4.1 from the literature. The attributes of attacker are chosen subjectively and are given in Table 4.1.

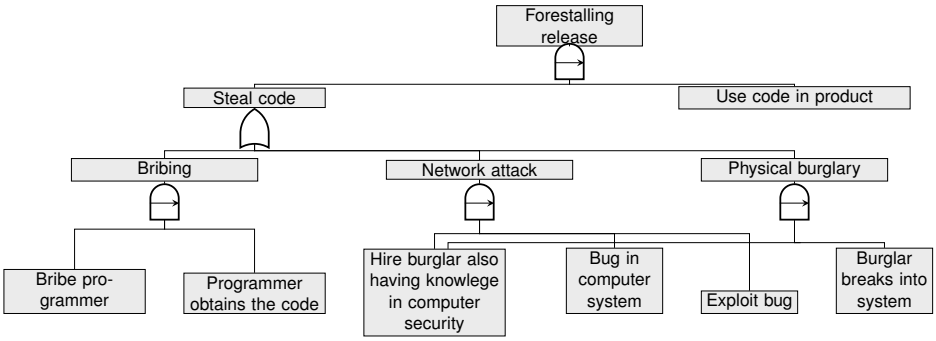

Figure 4.1.: Example of attack tree for forestalling release of software

Example of an attack tree. Fig. 4.1 shows an attack tree taken from (Buldas et al., 2006a). The example has been modified by adding sequential temporal dependencies. Here, a competitor steals a piece of software code and then builds it into his own product, as modelled by the top-level SAND gate. The OR gate at node *Steal code* shows that the code can be stolen in three different ways: via *Bribing*, a *Network Attack* or *Physical burglary*. Bribing is modelled as a two-step sequential process of first successfully bribing a programmer and then obtaining the code, represented by a SAND-gate.

Similarly, one can employ a burglar who has knowledge of computer security. This prerequisite though seems vague, is taken intentionally to show that our analysis tools can handle shared leaves/ sub-trees. Thus, this atomic step – "Employ burglar having also having knowledge of computer security" can lead to two different attack paths modelled through a shared node. One in which the hired person finds a bug and exploits it to obtain the code via a network attack, and another path in which he is physically involved in a burglary after being hired to steal the code. This dependency is again modelled through a SAND gate.

	Attacker		Values		
BAS	Profile	Skill	Time (in days)	Cost (in US $)	Cost to company (in US $)
Bribe a programmer	Generic attacker	Low	15-20	1500 + 50t	500.000
	Generic attacker	Med	10-20	1000 + 150t	500.000
	Generic attacker	High	0-10	500	500.000
	Software Engineer	Any	0-5	5000 + 100t	500.000
Programmer obtains the code	Generic attacker	Any	5-15	1000 + 100t	1.000.000
	Software Engineer	Any	0-5	2000 + 50t	1.000.000
Hire burglar with knowledge of computer security	Any	Any	5-15	4000 + 50t	0
Bug in Computer system	Any	Low	15-20	1000 + 50t	0
	Any	Med	5-10	1000 + 50t	0
	Any	High	0-5	1000 + 50t	0
Person exploits the bug	Any	Any	0-5	1000 + 50t	1.000.000
Person breaks into the system	Any	Any	0-5	2000 + 100t	400.000
Code is completed into product	Any	Any	5-15	2000 + 50t	100.000

Table 4.1.: Values used for annotating leaves of the attack tree as in figure 4.1

The process of constructing of attack trees and then extracting stochastic models is currently manual. It is foreseen that both these steps will be automated in the near future within the life cycle of project.

4.1.4. Analysis

Instead of translating the entire attack tree to the lower order formalism of priced timed automata in one step, we compose the adjacent attack steps based on their action labels iteratively and minimize them using compositional aggregation techniques such as strong bisimulations as in Figure 4.2. This modular approach yields a flexible framework that can easily be extended with future needs, such as countermeasures.

We express our security questions in an extension of the *Weighted CTL* logic (Brihaye, Bruyère, & Raskin, 2004), over a PTA whose locations l are decorated with a set of atomic propositions *Prop*$(l) \subseteq AP$. We slightly extend the syntax given by (Bouyer, 2006): Rather than providing a single constraint $v \sim c$ asking that value v meets bound c, we need a vector of constraints $x_i \sim c_i$, asking that all values v_i meet their bounds c_i. Thus, we can find the security metrics as enumerated in section 4.1.1 as follows:

(Unconstrained) attack values and attack paths: The UPPAAL CORA program has a built-in method to find an optimum if only one cost needs to be tracked. Here, we obtain the *Optimal accumulated costs* through the 'Best first' function built-in UPPAAL CORA.

(Constrained) attack values and attack paths: Optimal attack values can be obtained by repeatedly querying for the existence of traces reaching the attack goal with increasingly tight constraints. When the tightest possible bound has been obtained, this corresponds to an optimum. Since a positive result for the query also produces a trace that satisfies it, this procedure also yields an optimal attack path.

For example, to obtain the minimum time to succeed in the AT in Figure 4.1 given a cost limit of 10000 (assuming this is the only cost variable), we first query $\exists\Diamond_{x_{\mathsf{Top}}\geq 0, C\leq 10000}(P_{\mathsf{Top}}.Goal)$ to obtain some successful attack and its corresponding time, e.g. suppose this yields an attack that takes 10 days to complete, then we perform a new query $\exists\Diamond_{x_{\mathsf{Top}}< 10, C\leq 10000}(P_{\mathsf{Top}}.Goal)$ to try to find a faster attack. If no such attack exists, we know that the minimal time to complete an attack given the budget is 10 days, and we have obtained an attack path that succeeds in this time and budget.

Ranking: To find different attacks ranked according to their cost, we repeat the procedure above, each time excluding the attack paths we have already found. For example, if the attack consisting of BASs 1 and 3 is the fastest possible attack, the second-fastest is found using the query $\exists\Diamond_{x_{\mathsf{Top}}\leq T}(P_{\mathsf{Top}}.Goal \wedge \neg(P(v_1, A).Success \wedge P(v_3, A).Success))$ and finding the smallest value for T to obtain the second-fastest attack. This process can be repeated until the desired number of optimal attacks has been found.

Pareto optimal curves: Pareto optimal curves can be obtained by finding the optimal attacks subject to an increasing constraint. For example, to find the curve of minimal time vs. cost, we begin by finding the minimal time to attack, and computing the lowest-cost

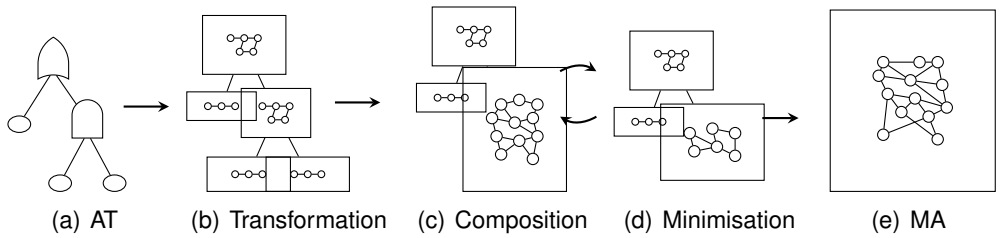

Figure 4.2.: Graphical overview of compositional aggregation for AT models.

attack that meets this time bound. Then, we compute the minimal time to attack with a smaller budget, and again find the lowest-cost attack that meets the new time bound. This process is repeated until no attacks exist that meet the latest budget.

To illustrate, consider again the attack tree in Figure 4.1. The minimal time to complete an attack is 5 days, and the lowest-cost attack that meets this time limits costs $9250. The fastest attack that costs less than $9250 takes 10 days, and the lowest cost attack that can be performed within 10 days costs $8500. There is no attack that costs less than $8500. Thus, we obtain the pareto curve shown in Figure 4.4.

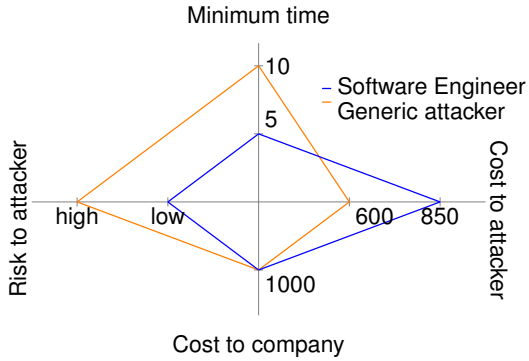

Figure 4.3.: Optimal Resources(Time/Cost) for Generic Attacker/Software Engineer .

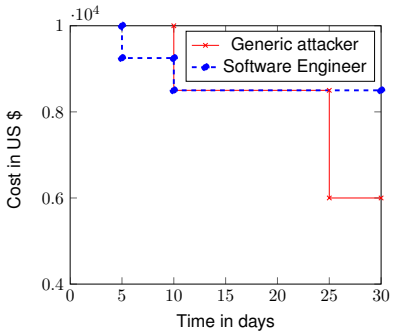

Figure 4.4.: Pareto curve of attack tree in Figure 4.1

Results In Figure 4.4, we see that both the attack values and the choice of attack path heavily depend on the attacker profile. In contrast to the generic attacker whose cost optimal attack trace is to bribe a programmer, a better skilled software engineer exploits a bug in the computer system to steal the code. The minimum time required to accomplish the attack also heavily depends on which attack steps are executed and when.

While a generic attacker takes 10 days to successfully execute the attack by physical burglary, a software engineer with insider benefits takes only 5 days to accomplish his goal. Also, there is an attack trace i.e *Hire a burglar, Burglar breaks into system, Code is completed into product* which results in an *optimum Cost to company* as $500,000 irrespective of the considered attacker profiles. The analysis results are presented as

a Pareto curve in Figure 4.4, where the generic attacker requires 10 days incurring a minimum cost of $9250 while a software engineer incurs a cost of $8500, but can complete the attack in 5 days.

Figure 4.3 provides a succinct representation of these different attack scenarios. We put the attacker objectives as vertices (i.e Minimum cost, Minimal time, Risk appetite, Cost to company) and the connecting lines are the attacker profiles. The figure shows a trade-off among different attack values for the considered attacker profiles which an enterprise risk manager can use to effectively plan countermeasures.

4.1.5. Conclusion

Priced timed automata provide a good representation of the attacker model. By providing insights on system vulnerabilities through several dimensions, our analysis can be used by system designers to make well informed decisions in selecting countermeasures and in prioritising their security investments.

4.2. Quantitative Attack Tree Analysis via Markov automata

4.2.1. Motivation

We use the Markov automata formalism to model temporal and causal dependencies (also refer section 2.2.2) in ATs, and provide insights on (1) static probabilistic questions such as probability to reach goal, (2) time-dependent questions such as probability of success as a function of time and (3) comprehensive questions from both models. Concretely, we can answer the following security questions:

1. Given a probability distribution of attack execution time and eventual probability of success of atomic attack steps, what is the probability that system is compromised within time t?
2. How much time does an attacker need with a given success percentage to reach his goal?
3. Which basic attack step has the most impact in the attack tree? Putting it in context of temporal analysis, we can answer “The execution time of which BAS impacts the total execution time of the whole attack scenario most significantly?”

Thus, our framework of analysis is used to model the stochastic behaviour of the attacker and is based on model checking to derive precise values by providing a formal and fully compositional semantics.

4.2.2. Methodology and model

The representations of attack trees with Markov automata (The TRESPASS Project, D3.2.1, 2015) brings about two strong advantages. First of all, the high expressiveness of the formalism allows to model any temporal and stochastic dependency in an attack. On the other hand, the resulting model can be analysed efficiently by exploiting state-of-the art compression and model checking algorithms.

Our model requires each BAS (atomic attack steps i.e. leaves of attack tree) to be annotated with a CDF (Cumulative distribution function) of the execution time as well as a probability of success. Generally, there are two ways to obtain this data: (1) with historical data; or (2) on the basis of expert opinion. If empirical data is available, we can derive estimates for both the probability of success as well as the average execution time t. We can then approximate the execution time with $\exp(1/t)$. This is justified by the fact that the exponential distribution has maximal entropy. Indeed, many attack models use the exponential distribution as a default choice. If there is no data available, experts estimate t and the probability of success.

Compositional aggregation. We exploit the compositional aggregation technique introduced for dynamic fault trees (DFTs) in (Boudali, Crouzen, & Stoelinga, 2010). The general idea is to have a modular framework, where each element of the AT is represented by the corresponding I/O-MA. These models interact through synchronisation on their input and output signals. Thus, we compose the I/O-MA models of the AT elements to obtain an I/O-MA which represents the whole AT. In general, the state space of composed models grows exponentially in the number of states before the composition, a phenomenon known as state-space explosion. To combat this problem, we use a technique called *compositional aggregation.* That is, rather than composing the whole tree at once,

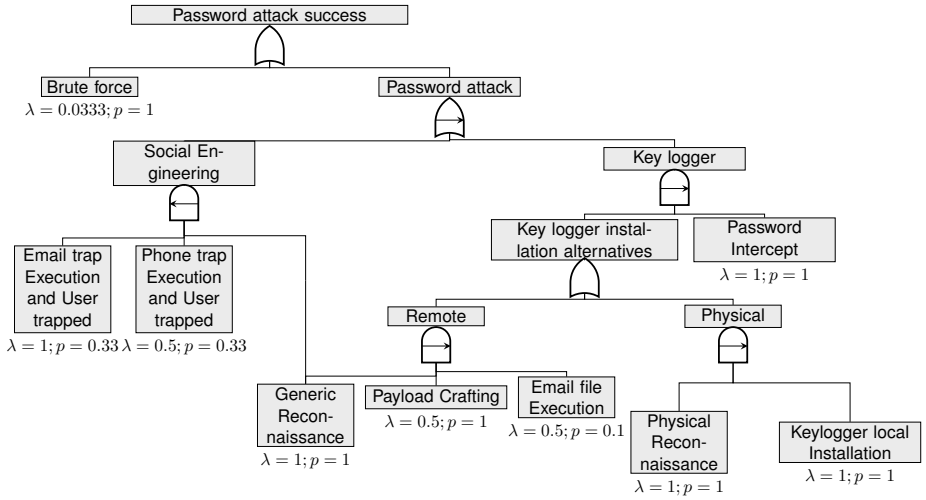

Figure 4.5.: Dynamic Attack Tree model: attack on password protected file.

we compose smaller sub-trees in a stepwise fashion and minimise the state space after each composition. A graphical representation of the approach is given in Figure 4.2.

Measures. The use of MAs allows us to define probability measures on an AT A in terms of its corresponding I/O-MA M_A. We define Success as the set of states in M_A in which the top event is reached. This enables us now to formally define the measures described in Section 4.2.1. The security metrics can be formally expressed as :

Static security Given an AT A, the probability of success of an attack in A in unlimited time is given by $\Pr_{M_A}(\Diamond \text{Success})$.

Time-dependent security Given an AT A and a mission time $t \in \mathbb{R}_{>0}$, the probability of success of an attack in A given t is given by $\Pr_{M_A}(\Diamond^{\leq t} \text{Success})$.

Mean time to attack (MTTA) Given an AT A the mean time of an attack in A is given by $\text{ET}_{M_A}(\Diamond \text{Success})$, where ET is defined as the expected time.

Node sharing. The communication between nodes is realised via the exchange of signals. This approach allows one particular node to communicate with more than one parent node at a time. Consequently, the semantics of our attack tree model can handle node sharing. Although this may eliminate the rigid tree structure, it opens new possibilities to model stochastic dependencies between different sub-trees in an AT.

Sensitivity analysis. To determine the effect of slight variances in the input data, one may ask "which BAS impacts the total execution time of the whole scenario most significantly?". We can answer this question by performing a sensitivity analysis and adapt the parameters of the CDF of a chosen BASs, leaving the others fixed. By comparing the results with the one from the original scenario we can then analyse in how far parameter changes of this BAS impact the result for the whole attack scenario.

4.2.3. Example: Attack of a password protected file

To obtain the security metrics as defined in section 4.2.2 and validate our methodology, we take a small case study directly from the literature. The following case study shown by attack tree, as in figure 4.5 is taken from (Piètre-Cambacédès & Bouissou, 2010). It is enriched by adding sequential gates to take care of temporal dependencies and has shared leaves, thus deviating from a pure tree formalism. Here, the goal is to obtain the password of a password protected file. There are two main ways to obtain it: (1) by cracking it through bruteforce, or (2) by trying to obtain the password.

We annotate each BAS with the mean time to execution (MTTE) defined by rate λ and a probability of success denoted by p. Those values are based on the intrinsic difficulty, available resources, estimated skills and the level of protection as provided in (Piètre-Cambacédès & Bouissou, 2010).

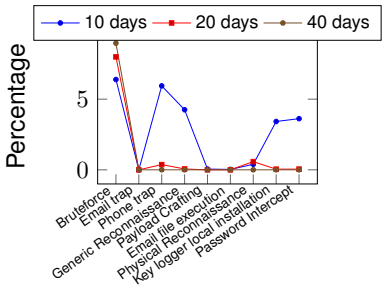

Figure 4.6.: Probability of success for case: Password protected file.

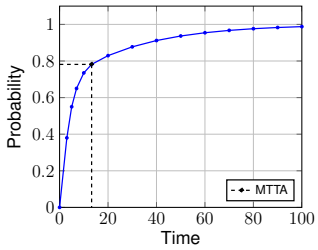

Figure 4.7.: Sensitivity analysis for case: Password protected file

Experimental results Figure 4.6 depicts the probability of a successful attack as a function over time. Our analysis results show that the attacker succeeds with about probability of success of 61.9% after one week while the mean time of a successful attack is 13.2 days. To find, which attack step execution has the most serious impact on system security, we perform a sensitivity analysis (Ou & Dugan, 2000). Here, we run the analysis multiple times and in each run, we slightly change one of the BAS attributes while keeping the others fixed. Then, by ranking BAS in ascending order of percentage change in execution time, we can argue which BAS has the most impact on the execution time of an attack tree in a temporal context. The added benefit of doing sensitivity analysis is that we can figure, how much the output is vulnerable to input deviations.

Thus, by doing sensitivity analysis we can conclude that both the BASs *Phone trap execution with user trapped* and *Generic reconnaissance* remain equally important in the short term, i.e. 7-10 days. Figure 4.7 reveals that BAS– *Infection of Control PC*, *Intercept in/out signals* and *Collect data* have the greatest impact in the short and long term. To obtain these results, all experiments were computed on an Intel Xeon CPU E5335 at 2.00 GHz with 22 GB RAM under Linux. The average runtime for one experimental run for each case study is 76.57 sec for the password protected file.

4.2.4. Conclusion

The timed dynamic analysis through Markov automata provides a security practitioner with an estimate of how an attacker is expected to behave temporally – can the attacks be executed successfully in hours, days or rather months? In the context of enterprise security it equips system architects with a tool to predict the likelihood of attack and thus make well informed decisions, by knowing which basic attack steps are more critical for success than others and which countermeasures should be prioritised in the short/ medium and long term.

4.3. Pareto Efficient Solutions for Attack Trees

4.3.1. Motivation

Most approaches model the attacker's behaviour on a systems by considering attack trees with one parameter and analysing a particular aspect of an attack, such as feasibility or cost. The analyses are performed by assigning values to the basic actions and traversing the tree from the leaves to the root. Different analytical methods suggested different functional operators for computing the value from child nodes to the parent node, based on the type of refinement. However, attack trees with one parameter might not be enough for capturing the attacker's behaviour in large and complex attack scenarios. Various extensions of attack trees with multiple parameters have been studied (refer Section 3.2). In most multi-parameter models, values characterising basic attacks are propagating to the root relying on local decision strategies. In case of incomparable values, however, this approach may yield sup-optimal results.

In order to tackle this challenge, we devise automated techniques that optimise all parameters at once. Moreover, in the case of conflicting parameters our techniques compute the set of all optimal solutions, defined in terms of Pareto efficiency.

This work is the complete version of the analysis technique "Pareto Frontier for Attack Trees", described in (The TREsPASS Project, D3.3.1, 2013).

4.3.2. Methodology and Model

Our model addresses multi-parameter optimisation of attack trees. We present evaluation techniques that characterise the leaves of a tree with more than one parameter, such as success probability *and* cost of an attack. Our techniques compute different aspects of the scenario and handle multiple parameters, thus optimising all of them at once. Multi-parameter optimisation becomes necessary in case of conflicting parameters, as there is no single best solution but rather a set of optimal solutions. We handle conflicting parameters by computing the set of efficient solutions, defined in terms of Pareto efficiency. Thus, Pareto efficiency handles the multi-criteria optimisation problem, as well as parameters with incomparable values.

In particular, we study the problem in the settings of a Boolean and a probabilistic semantics for attack trees. For each such semantics, we first consider the problem of feasibility of the attack, and then we extend our technique to compute optimal attacks in the presence of cost.

Standard attack trees combine sub-trees either conjunctively or disjunctively. We extend attack trees with negation operator. Such an operator allows to analyse a wider range of attack scenarios, including the cases of irreversible and conflicting actions, thus making trees more flexible and expressive. For instance, cutting a communication wire might be unrecoverable, and after having cut a wire a player might not be able to communicate with a given device. The negation operator is an additional feature, which is handy for

modelling purposes. However, presented analysis techniques are applicable for standard attack trees with only conjunction and disjunction operators as well.

4.3.3. Analysis

For evaluating the security properties of an attack scenario, we present two evaluate techniques of attack trees, termed semantic and algorithmic evaluation respectively.

The semantic evaluation of an attack tree analyses the tree by considering all possible Boolean assignments of values to the basic actions of the tree. It computes the value for the root node in a top-down fashion.

The semantic evaluation characterises the analysis in a natural way, for it explicitly considers all the interactions interwoven in a tree in terms of assignments to the leaves. Nonetheless, it gives rise to an exponential computation already in the Boolean case, the satisfiability problem being NP-complete. Therefore, evaluation techniques that enjoy a lower complexity are needed. In particular, we face the problem of defining those restrictions on attack trees under which the more efficient methods are sound with respect to the semantic evaluation, our gold standard. We improve dramatically on the complexity devising an algorithmic evaluation that is linear in the size of the tree and yet sound for an expressive sub-class of models.

The algorithmic evaluation of an attack tree considers the values of basic actions and propagates them up to the root by traversing the tree from the leaves to the root. The propagation is performed according to a standard bottom-up algorithm, using appropriate mathematical operators.

We show the equivalence of the two evaluations under the condition that no action occurs twice in the tree.

In the evaluations the computation of multiple conflicting parameters plays an important role. For instance, evaluating the pairs of success probability *and* cost values leads to a multi-parameter optimisation. Moreover, such pairs are incomparable in case the goal of an attacker is to maximise one parameter while minimising the other. In order to address multi-parameter optimisation in the case of incomparable values, we resort to Pareto efficiency and define functions for computing the sets of Pareto efficient solutions. A solution is called Pareto efficient if it is not dominated by any other solution in the ordering relation (Legriel, Guernic, Cotton, & Maler, 2010).

In order to compute the set of pairs of efficient solutions, where we want to maximise the first argument while minimising the second, we define a function that computes the set of all pairs that have higher value for the first argument *or* lower value for the second argument with respect to the other pairs in the set.

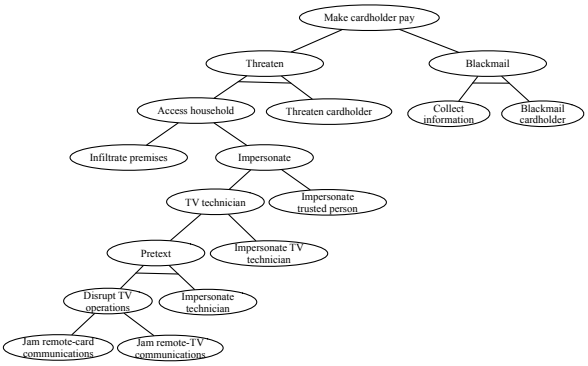

Figure 4.8.: Attack tree for forcing the card-holder to pay

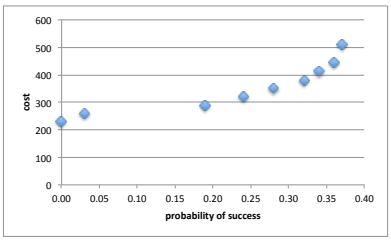

Figure 4.9.: Pareto optimal solutions.

4.3.4. Example

We demonstrate our evaluation techniques on a home-payment system. The system is specifically designed to help elderly or disabled people, who may have difficulties leaving their home, to pay for some services, e.g. care-taking or rent. The payment is performed using the remote control of a television box with a contactless payment card. When a transfer is initiated, a password is needed in order to authenticate the owner of the card.

The attack scenario that we consider is to steal money from the card-holder by forcing him/her to pay fake services. The corresponding attack tree is given in Figure 4.8. A more detailed explanation of the example, as well as the attack tree, is presented in (The TRESPASS Project, D3.3.1, 2013).

We decorate the basic actions of the attack tree in Figure 4.8 with values for probability of success and cost. In order to detect the attacks with maximum probability of success and minimum cost, we apply the algorithmic evaluation, i.e. we propagate the values of basic actions from the leaves to the root of the tree. The overall result of the evaluation, i.e., the set of efficient solutions for the goal (the root) representing the Pareto frontier of the problem, is displayed in Figure 4.9. The probability of successful attacks ranges from 0 to 0.37 and the corresponding cost ranges from 0 to 510. The intermediate points on the Pareto frontier indicate other optimal solutions. We can conclude that the system under study is (p, c)-vulnerable for all the incomparable pairs in the Pareto frontier, where p is probability and c is cost.

4.3.5. Extension with Countermeasures

Attack trees evaluate only the attacker's behaviour and do not consider possible defences undertaken to avoid the attacks. To overcome this limitation, further extensions of attack trees for capturing the defender's behaviour have been studied (refer to 2.1).

Similar to attack tree analyses, most analyses of attack-defence trees focus on one specific aspect of the system and do not consider multiple parameters and the subsequent need for optimising all of them at once.

In order to tackle this issue, we have extended our attack tree evaluation techniques to attack-defence trees. Below, we give a short explanation of the evaluation techniques. For the technical development we refer the reader to (Aslanyan & Nielson, 2015).

We have presented evaluation techniques for multi-parameter attack-defence trees that optimise all parameters at once, leveraging the concept of Pareto efficiency, in (Aslanyan & Nielson, 2015). Our developments have been carried out on a new language-based formalism for attack-defence trees, which extends standard trees with negation and with a novel operator for player alternation. The player alternation operator changes the goal of a sub-tree by changing the type of the player. For instance, if the sub-tree belongs to an attacker with the corresponding goal (e.g., minimising), then the tree belongs to a defender with the corresponding goal (e.g., maximising). Thus, the player alternation operator flips the player from an attacker to a defender and vice versa. In this language, the interaction between an attacker and a defender is made explicit by associating a player to each node thanks to a simple type system.

We have developed analyses of attack-defence scenarios both in the Boolean and in the probabilistic settings, investigating aspects such as the feasibility and the cost of an attack or a defence. For each case we have illustrated the natural semantic evaluation technique as well as an algorithmic evaluation which enjoys a dramatic improvement in complexity, and we have proven under which conditions the latter can be relied on in place of the former. Both methods characterise the goal of the scenario with a set of Pareto efficient solutions.

4.3.6. Conclusion

We developed analysis technique for evaluating attack trees will multiple parameters. The evaluation optimises all parameters at once and computes the set of optimal solutions in terms of Pareto efficiency. We extend further the model with countermeasures and provide an evaluation technique for attack-defence trees with more than one parameter for basic actions.

The analysis technique takes as an input an attack tree (attack-defence tree respectively), which is generated either automatically or by hand, and values for basic actions, such as probability of success and cost. The technique computes the Pareto set of optimal solutions and provides the set of basic actions leading to the successful attack.

4.4. Statistical Model Checking of Timed Automata

4.4.1. Motivation

Model checking is a technique that has a long tradition in formal methods and verification. It has been sucessfully applied to a wide variety of problems, including many with safety- and/or security-critical concerns. In spite of the success of (traditional) model checking and the powerful formal models underlying them, it has been shown that these are not expressive enough to adequately capture the complex and intricate behaviour of *cyber physical systems*. Such systems often exhibit rich dynamics and stochastic behaviour. In fact, it has been shown that the model checking problem for such systems is undecidable and therefore only approximate solutions can be found using traditional model checkers.

To overcome this problem, *statistical model checking* has been proposed. The essential idea of statistical model checking is to monitor a (large) number of runs of the (model of the) system and then use advanced statistical tools and theories, such as sequential hypothesis testing or Monte Carlo simulation, to estimate if the system satisfies a given property and give a confidence interval for such an estimate. Properties are typically specified in LTL or MITL and the number of runs depends on the desired level of precision and confidence of the results. As such, statistical model checking represents a compromise between testing and traditional model checking techniques. Furthermore, the technique does not suffer from the so-called state-space explosion problem that plagues classical model-checking methods and scales much more efficiently with the size of the model.

Due to the nature of SMC (simulation rather than model checking), more complex systems can be modelled and even undecidable problems can be approximated. The use of advanced statistical techniques, such as sequential hypothesis testing, combined with the opportunity to distribute the simulations runs, makes statistical model checking higly efficient.

For a thorough introduction to statistical model checking and the underlying theory, we refer to the UPPAAL SMC tutorial (David, Larsen, Legay, Mikucionis, & Poulsen, 2015).

4.4.2. Methodology and Model

The formalism underlying our approach to statistical model checking is an extension of the well-known theory of *timed automata* (Alur & Dill, 1994b) with a *stochastic interpretation* called *stochastic timed automata* (David et al., 2015). Specifically, in a traditional timed automata model, non-deterministic choice is used to resolve situations where several actions (transitions) may be enabled at the same time. In our stochastic interpretation, non-deterministic choice is replaced with probabilistic choices and non-deterministic time delays are replaced with probability distributions: either uniform distributions (for bounded delays) or exponential distributions (for unbounded delays).

Recent work has explored the possibility of further extending the theory of stochastic timed automata with ability to *dynamically* create new processes. This extension significantly

increases the expressive power of the formalism and can not be handled (directly) using traditional model checking techniques.

Based on this formalism of stochastic timed automata, we can apply the technique of statistical model checking for analysing both qualitative and quantitative properties. We describe this in more detail below.

4.4.3. Analysis

Statistical model checking, the primary mode of analysis for stochastic timed automata, can be seen as a combination of traditional (qualitative) model checking with facilities for simulation and (statistical) testing. In addition to the usual (qualitative) queries verified by traditional model checking, e.g., reachability, invariance, and leads-to, statistical model checking extends the range of possible queries to include probability estimation, hypothesis testing, and probability comparison.

The UPPAAL SMC tool, part of the UPPAAL tool suite, is a state-of-the-art statistical model checker for stochastic timed automata (David et al., 2015). The UPPAAL tool suite is a platform for modelling, simulation, and verification (through model checking) of systems that can be modelled as networks of timed automata, i.e., as a set of possibly non-deterministic processes with finite control and real-valued clocks that can communicate either over channels or through shared variables. One notable feature of UPPAAL SMC is that the formalism is extended to cover *hybrid systems* by allowing clocks in the underlying timed automata with rates given by general expressions, effectively using *ordinary differential equations*.

4.4.4. Conclusion

In summary, statistical model checking is a relatively novel formal technique, based on stochastic timed automata and combining traditional model checking with statistical testing theory and simulation. This combination yields a powerful method for both qualitative analysis and quantitative analysis. This makes statistical model checking an obvious candidate for use in the TRE$_S$PASS project.

4.5. Improved Failure-Free Model Analysis

4.5.1. Motivation

To the best of our knowledge, systematic and scientific approach to security engineering does not exist yet, but such an approach exists in other engineering areas – for instance, in civil engineering. Designing a new building or construction engineers need to make sure that it will not break. It is possible to calculate exact stress values which a construction will be experiencing during exploitation taking numerous conditions into account by solving

equations containing thousands of variables, but this approach cannot be called trivial. In order to verify that the construction will not break engineers calculate the upper bound of stress at which the construction definitely will not break, and they are not interested in complex calculations of the exact value of stress at which the construction breaks. It is desirable to have a simple and reliable method to calculate if the analyzed organization is secure, similar to the one used in civil engineering.

The improved failure-free model analysis, described in Section 4.5 of the deliverable D3.3.2, focuses on analyzing the security of organizations against targeted profit-oriented attacks executed by rational attackers. If the security assessment assumes rational adversarial behavior – e.g. assumes that a rational attacker will not attack if it is not profitable for him – every model, which tries to calculate exact adversarial outcome, will always contain a margin for an error and thus the entire attacker model fails. The reason for this is that in the case of the exact result, possible errors may propagate in both directions – in the direction of the lesser values, as well as in the direction of the greater values. Hence, the computational methods, which calculate the exact result, are capable of producing false-positive results – e.g. when the analysis shows that an organization is sufficiently protected, but in reality profitable attack vectors may still exist if, for instance, the operational security risk quantitative annotations are overlooked. A reliable computational method does not need to calculate the exact result, but approaches the result from above thus calculating its upper bound. The upper bound is reliable as its possible error margin extends only in one direction – in the direction of lesser values. It can be seen that such an approach does not produce false-positive results. In a reliable model the security assessment is based on the upper bound analysis and the corresponding computational methods calculate the upper bounds as the result.

Compared to the existing models (Buldas, Laud, Priisalu, Saarepera, & Willemson, 2006b; Jürgenson & Willemson, 2008, 2010, 2009; Buldas & Stepanenko, 2012b), the improved failure-free model proposes a more reliable model and more efficient computational methods for finding upper bounds of the expected outcome of a rational attacker, considering the so-called fully-adaptive adversarial setting in which the attacker is able to run atomic sub-attacks in arbitrary order. In the improved failure-free model there always exist optimal attacking strategies that are non-adaptive in the sense that the next tried sub-attack does not depend on the results of the previously tried attacks. Due to the simplicity of the optimal strategies in the improved failure-free model, upper bounds for adversarial utility can be found in near-linear time, and hence the model is especially suitable for creating simple and reliable engineering methods in information security.

4.5.2. Methodology and Model

The attacker game in the improved failure-free model can be described as a single-player game played by the adversary. Every state in the game is represented by a monotone Boolean function $\mathcal{F}$, where each input variable x_i in $\mathcal{F}$ is annotated with success probability p_i and expenses $\mathcal{E}_i$ which is modeled as a random variable. In the initial state the game corresponds to the Boolean function $\mathcal{F}$ equivalent to the attack tree containing all possible moves of the adversary in the game. In this setting the adversarial goal is to satisfy

$\mathcal{F}(x_1, \ldots, x_k)$ by picking variables x_i one at a time and assigning $x_i = 1$. Actions that the adversary undertakes in every instance of the game can be described as follows:

1. The adversary executes a move x_i and pays a certain amount of expenses $e_i \leftarrow \mathcal{E}_i$. This random choice is independent from other random choices from $\mathcal{E}_i$.
2. With a certain probability p_i the move x_i succeeds and the function $\mathcal{F}$ representing the current game instance is transformed to its derived form $\mathcal{F}|_{x_i=1}$, where by $\mathcal{F}|_{x_i=1}$ we mean the function $\mathcal{F}(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_m)$ derived from $\mathcal{F}$ by the assignment $x_i = 1$.
3. With probability $1 - p_i$ the game remains the same and its Boolean function does not change: $\mathcal{F}|_{x_i=0} \equiv \mathcal{F}$ due to the ability of the adversary to re-run failed attacks again an infinite number of times.

The game goes on until one of the following conditions is satisfied:

- $\mathcal{F} \equiv 1$ – the adversary satisfies $\mathcal{F}$, wins the prize $\mathcal{P}$ and thus wins the game
- $\mathcal{F} \equiv 0$ – the adversary fails to satisfy $\mathcal{F}$ and thus loses the game
- when the adversary decides to discontinue playing the game

4.5.3. Analysis

The analysis runs in two steps. First, for each of the common moves the expenses reduction technique is applied, followed by the expenses propagation method which computes the lower bound of adversarial expenses $\mathcal{E}(\mathcal{G})$ in game $\mathcal{G}$. The upper bound of adversarial utility $\mathcal{U}(\mathcal{G})$ may be computed as $\mathcal{P} - \mathcal{E}(\mathcal{G})$, where $\mathcal{P}$ is the prize of the game. The idea behind expenses reduction is the substitution of every occurrence of a common move x with independent variable x'_i such that $\sum_{i=1}^{r}(x)x'_{x_i} = w(x)$, where $w(x)$ is the weight of x and $r(x)$ is the number of times that x is present in $\mathcal{G}$. Then the lower bound of adversarial expenses in $\mathcal{G}$ may be computed using algorithm 4.5.1 according to (4.1).

$$\begin{aligned} \mathcal{E}(\mathcal{G}_1 \wedge \ldots \wedge \mathcal{G}_k) &= \max\{\mathcal{E}(\mathcal{G}_1, \ldots, \mathcal{G}_k)\} \quad . \\ \mathcal{E}(\mathcal{G}_1 \vee \ldots \vee \mathcal{G}_k) &= \min\{\mathcal{E}(\mathcal{G}_1, \ldots, \mathcal{G}_k)\} \quad , \end{aligned} \tag{4.1}$$

where $\mathcal{G}_1, \ldots, \mathcal{G}_k$ are the sub-games of game $\mathcal{G}$ played by the adversary.

4.5.4. Genetic Approximations

The genetic approach tries to optimize the adversarial expected utility over the set of feasible solutions. Individuals (satisfying assignments to the monotone Boolean function of the game) are represented in linear binary form to facilitate the robustness of the crossover and mutation operators. The computational procedure used to generate individuals is shown in Algorithm 4.5.2. We allow duplicate entries in the population for the

ALGORITHM 4.5.1: Iterated *Expenses* propagation

Input: The game $\mathcal{G}$
Output: The expenses of the game $\mathcal{E}(\mathcal{G})$ (a real number)

```
1  Procedure ComputeExpenses (G)
2  if m is a conjunctive game instance then
3      expenses := 0
4      forall the sub-games i of m do
5          expenses += ComputeExpenses (i)
6  else if m is a disjunctive game instance then
7      cheapest := FindCheapestSubGame (m)
8      return cheapest
9  else m is an atomic move
10     return E (m)
```

sake of maintaining genetic variation and keeping the population size constant throughout the reproduction process. Genetic variation directly influences the chances of premature convergence and thus maintaining genetic variation in the population is one of the design goals. For the genetic algorithm application in the improved failure-free satisfiability games, the optimal population size is at least 180% of the size of the attack tree (the number of leaves in the attack tree). For the adaptive generic algorithm this value is 200% (Lenin, Willemson, & Charnamord, 2015). The fitness function for a given individual $\mathcal{I}$ in the failure-free model is defined by equation (4.2).

Algorithm 4.5.2: Recursive individual generation algorithm

Data: The root of a propositional directed acyclic graph (PDAG) representing a monotone Boolean function. An empty individual with all bits set to 0.
Result: Live individual.

```
1  if the root is a leaf then
2      get the index of the leaf;
3      set corresponding individual's bit to 1;
4  else if the root is an AND node then
5      forall the children of the root do
6          recursive call: child considered as root parameter;
7  else if the root is an OR node then
8      choose at least one child;
9      forall the chosen children do
10         recursive call: child considered as root parameter;
```

$$\mathcal{U}(\mathcal{I}) = \mathcal{P} - \sum_{x_i \in \mathcal{I}} \frac{\overline{\mathcal{E}_i}}{p_i} \ . \tag{4.2}$$

In the current implementation of the genetic algorithm for the failure-free satisfiability games the parent selection has been disabled entirely thus defaulting to crossing every individual with every other individual in the population (crossover rate equal to one), as scalable selection pressure comes along with the selection mechanisms after reproduction. The uniform crossover type has been chosen as it enables a more exploratory approach to crossover than the traditional exploitative approach, resulting in a more complete exploration of the search space with maintaining the exchange of good information. The crossover operator is shown in Algorithm 4.5.3. The uniform mutation type has been chosen for the improved failure-free model and the mutation rate was set to 0.1.

Algorithm 4.5.3: The uniform crossover operation

Data: The population of individuals represented as a sorted set.
Result: The population with new added individuals, created during the crossover operation.

```
1 initialize a new set of individuals;
2 forall the individual i in the population do
3     forall the individual j different from i do
4         new individual := the result of cross operation between individuals i and j;
5         if new individual is alive then
6             add the new individual to the set of new individuals;
7 add the set of new individuals to the population;
```

Thus, the following set of control parameter may be used to apply genetic algorithm to the domain of the improved failure-free models:

- Crossover operator: uniform crossover.
- Crossover rate: 1.
- Selection operator: missing.
- Mutation operator: uniform mutation.
- Mutation rate: 0.1.

The genetic algorithm allows to analyze attack trees with approximately 800 leaves in reasonable time, set to two hours. For the details we refer the reader to (Lenin et al., 2015).

4.5.5. Case Study/Example

In this section the improved failure-free analysis will be executed to analyze the security of the Estonian i-voting system (Heiberg & Willemson, 2014). The paper outlines the i-voting system, its structure, design considerations, as well as introduces an attack tree which describes possible ways how an attacker can affect the result of the elections. The corresponding attack tree may be found in http://research.cyber.ee/~antonc/ATA/

Voting_Attack_Tree.xml. The improved failure-free model analysis results are the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<adtree profit="989784.62"/>
```

As the result says that the adversarial utility is positive which means that there are profitable ways to attack the system, a more thorough analysis is needed. Executing the adaptive genetic analysis we obtain the most profitable attack vector outlined in Appendix B. It can be seen that in order to successfully attack an i-voting system the attacker needs to execute the revocation attack during which he distributes a fake voting application, tricks voters into using it, and then convinces the National Election Committee that the voting results are not legitimate.

4.5.6. Conclusion

The Improved Failure-Free model (Buldas & Lenin, 2013) is a representative of a Failure-Free SAT game. The input for the model is an attack tree in the form of a PDAG representing a monotone Boolean function $\mathcal{G}$ with leaves annotated with quantitative parameters for *Cost* and *Probability of success*, and a global parameter *profit* set for the entire attack tree. The computational methods compute the upper bound of adversarial utility. If the resulting utility is zero – it is safe to conclude that the system is secure against rational profit-oriented adversaries. If the utility is positive, profitable attack vectors are likely to exist and more thorough analysis is required.

4.6. Risk Reduction Factor

4.6.1. Motivation

In TRICK Service, in order to quantify the risk specificity for the generation of the RRF (Risk Reduction Factor), we apply risk specificity to three elements:

- For each risk scenario, we determine if it specifically relates to confidentiality, integrity or availability, if it is of intentional, accidental or environmental causes, etc.
- For each security measure, we qualify their influences on every security criteria
- For assets, we directly define the influence of each security measure on each asset

The risk specificity step has partially already been performed by the auditors and the developers of the tools and specificity values are freely available for the security measures of ISO 27001 and ISO 27002, and the asset type, limited to 10 elements: Service, Information, Software, Hardware, Network, Staff, Immaterial, Business, Financial, and Compliance. The only elements which require defining during a risk analysis are the risk

specificity of risk scenarios added by the user and the risk specificity of customised security measures not covered by the ISO 27001 and the ISO 27002 standards.

An RRF is thus a coefficient representing the ALE reduction generated by complete implementation of the security measure.

4.6.2. Methodology

The computation of the RRF is as follows:

$$RRF = Assettype/Measure * Strength * Category * Type * Source * MaxRRF$$

4.6.3. Example

An example is available on the tools website: https://trespass.itrust.lu/

4.7. Performance Indicators

4.7.1. Motivation

We illustrate the use of performance indicators in the context of patch management.

The patch management observing utility regularly checks whether any updates are available for the operating system it runs on. Although the process described below is system-independent on principle, for the sake of illustration this document focuses on Debian operating systems and the APT package managing utility.

4.7.2. Methodology

The utility works as follows:

1. Get a list of all available updates;
2. For each update, fetch release date and associated CVE information (if available);
3. Deduce implementation rate (of patch management) from CVE scores and time passed since the patch releases;
4. Report implementation rate to the TRICK Service API.

Deduce the implementation rate: For each available update, the tool retrieves the (CVSS) score of the associated CVE if possible (otherwise a value of 0.0 is assumed). The score is a floating point number between 0 and 10 which correlates with the criticality of the vulnerability according to the following table.

Range	Criticality
0-3.9	Low
4-6.9	Medium
7-10	High

Table 4.2.: Criticality by CVE score

The utility assumes a direct dependency between the CVE score s and the deadline $D(s)$ by which a patch must be applied; more precisely, it uses the following formula.

$$D(s) = 30 * e^{-\frac{s}{3}} \text{ (in days)}$$

The motivation behind the use of an exponential function is that it matches best the following reaction times, which result from estimation and best practices.

CVE Score	Patch must be applied
10	after 1 day
7	after 3 days
4	after 7 days
0	after 30 days

Table 4.3.: Patch application deadline by CVE score

Given a deadline D, a preliminary implementation rate $IR(D, \Delta t)$ is computed, where Δt stands for the time passed since the patch was released. The latter follows the law:

$$IR(D, \Delta t) = 1 - \frac{1}{10} * 5^{(\Delta t - D)}$$

The motivation is given by the following considerations:

Criterion	Mathematical equivalent
The IR should start at (roughly) 100%	$IR(D, 0) \approx 100\%$
At the deadline the IR should have dropped to 90%	$IR(D, D) = 90\%$
After the deadline, the IR should rapidly drop to 0%.	$R(D, .) \sim exp$

Table 4.4.: Criteria imposed to temporal evolution of implementation rate

The final implementation rate of patch management is defined to be the minimum of all preliminary implementation rates (of all available updates). Indeed, the lowest IR value should determine the final IR, rather than the average value.

4.8. Comparison of TRESPASS Analysis Techniques

The techniques presented above cover a wide range of analysis approaches and allow to answer an extensive amount of questions a security practitioner might be interested in. The possible questions of interest are:

1. Which attack succeeds with maximum probability?
2. Which attack path has the lowest costs?
3. Which attack path has the maximum probability and the lowest cost?
4. How much time does an attacker need with a given success percentage to reach his goal?
5. What is the minimal time needed to complete a successful attack within a given budget?
6. What is the maximal damage that can be incurred in one year?
7. Given a certain attacker profile, which attack will he choose most likely?
8. Given a certain attacker profile, which is the minimum time or skill level needed to complete a successful attack?

To answer these question, we have developed techniques that either extend previously proposed techniques, or are completely new. Table 4.5 summarises and compares the key features characterising the techniques described in this document. It gives an overall idea about the strengths and weaknesses of each technique, and which can be used for what purposes.

Table 4.5.: Comparative table summarizing the presented quantitative analysis techniques

Technique	Input	Attributes	Output	Tool	Complexity
Time-dependent analysis with Markov automata	AT, a set of atomic attacks annotated with a distribution of the execution time and the probability of success	execution time of the attacker/ invested resources, probability of success	timed probability distribution of the execution time	DFTCalc	
Improved Failure-free model	AT annotated with quantitative attributes	cost-success ratio of attack, prize	utility upper bound	AttackTree Analyzer	linear in the case of independent attack trees, exponential (in the worst case) for DAGs
Pareto efficient solutions for ATrees	AT, parameter values for basic actions	feasibility, probability of success, cost	success of AT, maximum probability, Pareto frontier for probability and cost	ATE	The semantic evaluation is exponential and the algorithmic evaluation is linear in the size of a tree
Multiparameter optimization for attack tree using Priced timed automata	AT, attacker persona attribute values for executing basic actions	Budget and time constraints, skill level of attacker	Pareto frontier, Optimal cost, time to reach goal, Constrained cost within certain time to reach goal		
Risk Reduction Factor	Risk scenarios, security measures, assets	probability, cost	Risk Reduction Factor	TRICK Service	
Performance indicators	List of installed packages		Implementation Rate	TRICK Service	

5. Conclusions

This deliverable documented the stochastic analysis methods developed in WP3. This is the final deliverable of Task T3.3.

The deliverable covers a handful of techniques which can be used to evaluate attack scenario with respect to a number of different metrics. We introduced attack trees and other models used for analysing the security of an attack scenario. Attack trees are widely used technique for evaluating the security of the organisation. However, as there is not one best model that provides an inside of the attacker behaviour and characterise system vulnerabilities, some of the presented analysis techniques considered different models for their evaluation, such as Markov automata and priced timed automata. The extraction of these models is discussed in details in (The TRESPASS Project, D3.2.1, 2015).

This deliverable is the extension of the work presented in the deliverable (The TRESPASS Project, D3.3.1, 2013). In this deliverable we focused on new and improved analysis techniques. Some of the developed techniques covers the areas, such as

- Statistical model checking for analysis of complex models by simulation
- Pareto efficiency for optimising attack trees with multiple attributes
- Uppaal model checker for computing optimal traces

For each of these novel and improved techniques, we briefly outlined and motivated the underlying models and techniques in the core of this document. We give an overview of the key features and underlying ideas, and demonstrated that they can answer a wide variety of security-related questions. Finally, the techniques were compared with respect to their applications range and computational merits.

Several techniques, discussed in this deliverable, are supported by a tool. These tools are explained and presented through the videos, which are available in `https://vimeo.com/channels/trespass` (The TREsPASS Project, 2015).

References

Aijaz, A., Bochow, B., Dötzer, F., Festag, A., Gerlach, M., Kroh, R., & Leinmüller, T. (2006). Attacks on Inter Vehicle Communication Systems - an Analysis. In *3rd international workshop on intelligent transportation* (pp. 189–194).

Alur, R., & Dill, D. L. (1994a). A theory of timed automata. *Theoretical Computer Science*, *126*(2), 183 – 235. doi: 10.1016/0304-3975(94)90010-8

Alur, R., & Dill, D. L. (1994b). A theory of timed automata. *Theor. Comput. Sci.*, *126*(2), 183-235.

Aslanyan, Z., & Nielson, F. (2015). Pareto efficient solutions of attack-defence trees. In *Principles of security and trust - 4th international conference, POST 2015, held as part of the european joint conferences on theory and practice of software, ETAPS 2015, london, uk, april 11-18, 2015, proceedings* (pp. 95–114). Retrieved from http://dx.doi.org/10.1007/978-3-662-46666-7_6 doi: 10.1007/978-3-662-46666-7_6

Bagnato, A., Kordy, B., Meland, P. H., & Schweitzer, P. (2012). Attribute Decoration of Attack–Defense Trees. *International Journal of Secure Software Engineering (IJSSE)*, *3*(2), 1-35. doi: 10.4018/jsse.2012040101

Behrmann, G., Larsen, K. G., & Rasmussen, J. I. (2005a). Priced timed automata: Algorithms and applications. *Formal Methods for Components and Objects*, *3657*, 162 – 182. doi: 10.1007/11561163_8

Behrmann, G., Larsen, K. G., & Rasmussen, J. I. (2005b). Priced timed automata: Algorithms and applications. *Formal Methods for Components and Objects*, *3657*, 162 – 182. doi: 10.1007/11561163_8

Boudali, H., Crouzen, P., & Stoelinga, M. (2010). A rigorous, compositional, and extensible framework for dynamic fault tree analysis. *Dependable and Secure Computing, IEEE Transactions on*, *7*(2), 128–143.

Boudali, H., Haverkort, B. R., Kuntz, M., & Stoelinga, M. (2007). Best of three worlds: towards sound architectural dependability models.

Bouyer, P. (2006). Weighted timed automata: Model-checking and games. *Electronic Notes in Theoretical Computer Science*, *158*, 3–17. doi: 10.1016/j.entcs.2006.04.002

Brihaye, T., Bruyère, V., & Raskin, J. F. (2004). Model-checking for weighted timed automata. *Proc. FORMATS-FTRTFT '04*, *3253*, 277–292.

Buckshaw, D. L. (2014). *Use of decision support techniques for information system risk management*. John Wiley Sons, Ltd.

Buldas, A., Laud, P., Priisalu, J., Saarepera, M., & Willemson, J. (2006a). Rational choice of security measures via multi-parameter attack trees. In *Critical info. infrastructures security, first int. workshop, CRITIS, 2006* (pp. 235–248). doi: 10.1007/11962977_19

Buldas, A., Laud, P., Priisalu, J., Saarepera, M., & Willemson, J. (2006b). Rational Choice of Security Measures Via Multi-parameter Attack Trees. In J. López (Ed.), *Critical Information Infrastructures Security, First International Workshop, CRITIS 2006, Samos, Greece, August 31 - September 1, 2006, Revised Papers* (Vol. 4347, pp. 235–248). Springer. Retrieved from http://dx.doi.org/10.1007/11962977_19 doi: 10.1007/11962977_19

Buldas, A., & Lenin, A. (2013). New efficient utility upper bounds for the fully adaptive model of attack trees. In S. K. Das, C. Nita-Rotaru, & M. Kantarcioglu (Eds.), *Decision and game theory for security - 4th international conference, gamesec 2013, fort worth, tx, usa, november 11-12, 2013. proceedings* (Vol. 8252, pp. 192–205). Springer. Retrieved from http://dx.doi.org/10.1007/978-3-319-02786-9_12 doi: 10.1007/978-3-319-02786-9_12

Buldas, A., & Mägi, T. (2007). Practical Security Analysis of E-Voting Systems. In A. Miyaji, H. Kikuchi, & K. Rannenberg (Eds.), *IWSEC* (Vol. 4752, pp. 320–335). Springer. Retrieved from http://aeolus.ceid.upatras.gr/scientific-reports/2nd_year_reports/practical_e_voting_final.pdf

Buldas, A., & Stepanenko, R. (2012a). Upper bounds for adversaries' utility in attack trees. In J. Grossklags & J. C. Walrand (Eds.), *GameSec* (Vol. 7638, p. 98-117).

Buldas, A., & Stepanenko, R. (2012b). Upper Bounds for Adversaries' Utility in Attack Trees. In J. Grossklags & J. C. Walrand (Eds.), *Decision and Game Theory for Security - Third International Conference, GameSec 2012, Budapest, Hungary, November 5-6, 2012. Proceedings* (Vol. 7638, pp. 98–117). Springer. Retrieved from http://dx.doi.org/10.1007/978-3-642-34266-0_6 doi: 10.1007/978-3-642-34266-0_6

Byres, E. J., Franz, M., & Miller, D. (2004, December). The Use of Attack Trees in Assessing Vulnerabilities in SCADA Systems. In *International Infrastructure Survivability Workshop (IISW'04), Institute of Electrical and Electronics Engineers, Lisbon* (Vol. , p.). Retrieved from http://blogfranz.googlecode.com/files/SCADA-Attack-Trees-IISW.pdf

David, A., Larsen, K. G., Legay, A., Mikucionis, M., & Poulsen, D. B. (2015). Uppaal SMC tutorial. *STTT*, *17*(4), 397–415. Retrieved from http://dx.doi.org/10.1007/s10009-014-0361-y doi: 10.1007/s10009-014-0361-y

Eom, J.-H., Park, M.-W., Park, S.-H., & Chung, T.-M. (2011, February). A framework of defense system for prevention of insider's malicious behaviors. In *13th International Conference on Advanced Communication Technology (ICACT'11)* (pp. 982–987).

Guck, D., Hatefi, H., Hermanns, H., Katoen, J., & Timmer, M. (2013). Modelling, reduction and analysis of markov automata (extended version). *CoRR*, *abs/1305.7050*. Retrieved from http://arxiv.org/abs/1305.7050

Hartmanns, A., & Hermanns, H. (2015). In the quantitative automata zoo. *Science of Computer Programming*, -. Retrieved from http://www.sciencedirect.com/science/article/pii/S0167642315002580 doi: http://dx.doi.org/10.1016/j.scico.2015.08.009

Heiberg, S., & Willemson, J. (2014). Modeling Threats of a Voting Method. In *Design, Development, and Use of Secure Electronic Voting Systems* (pp. 128–148). IGI Global.

Henniger, O., Apvrille, L., Fuchs, A., Roudier, Y., Ruddle, A., & Weyl, B. (2009). Security

requirements for automotive on-board networks. In *9th international conference on intelligent transport systems telecommunications, (ITST)* (pp. 641–646). Lille. doi: 10.1109/ITST.2009.5399279

Hermanns, H. (2002). *Interactive markov chains: The quest for quantified quality* (Vol. 2428). Springer. Retrieved from http://dx.doi.org/10.1007/3-540-45804-2 doi: 10.1007/3-540-45804-2

Hoare, C. A. R. (1985). *Communicating sequential processes*. Prentice-Hall.

Jung, C., Elberzhager, F., Bagnato, A., & Raiteri, F. (2010, February). Practical Experience Gained from Modeling Security Goals: Using SGITs in an Industrial Project. In *International Conference on Availability, Reliability, and Security (ARES'10)* (pp. 531–536). doi: 10.1109/ARES.2010.12

Jürgenson, A., & Willemson, J. (2008). Computing Exact Outcomes of Multi-parameter Attack Trees. In R. Meersman & Z. Tari (Eds.), *Otm conferences (2)* (Vol. 5332, p. 1036-1051). Springer. Retrieved from http://dblp.uni-trier.de/db/conf/otm/otm2008-2.html#JurgensonW08

Jürgenson, A., & Willemson, J. (2008). Computing Exact Outcomes of Multi-parameter Attack Trees. In R. Meersman & Z. Tari (Eds.), *On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, November 9-14, 2008, Proceedings, Part II* (Vol. 5332, pp. 1036–1051). Springer. Retrieved from http://dx.doi.org/10.1007/978-3-540-88873-4_8 doi: 10.1007/978-3-540-88873-4_8

Jürgenson, A., & Willemson, J. (2009). Serial Model for Attack Tree Computations. In D. Lee & S. Hong (Eds.), *Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers* (Vol. 5984, pp. 118–128). Springer. Retrieved from http://dx.doi.org/10.1007/978-3-642-14423-3_9 doi: 10.1007/978-3-642-14423-3_9

Jürgenson, A., & Willemson, J. (2010). On Fast and Approximate Attack Tree Computations. In J. Kwak, R. H. Deng, Y. Won, & G. Wang (Eds.), *Information Security, Practice and Experience, 6th International Conference, ISPEC 2010, Seoul, Korea, May 12-13, 2010. Proceedings* (Vol. 6047, pp. 56–66). Springer. Retrieved from http://dx.doi.org/10.1007/978-3-642-12827-1_5 doi: 10.1007/978-3-642-12827-1_5

Kordy, B., Mauw, S., Radomirović, S., & Schweitzer, P. (2011). Foundations of Attack–Defense Trees. In P. Degano, S. Etalle, & J. D. Guttman (Eds.), *FAST* (Vol. 6561, pp. 80–95). Springer.

Kordy, B., Mauw, S., Radomirović, S., & Schweitzer, P. (2012). Attack–Defense Trees. *Journal of Logic and Computation*. (Available at http://logcom.oxfordjournals.org/content/early/2012/06/21/logcom.exs029) doi: 10.1093/logcom/exs029

Kordy, B., Mauw, S., & Schweitzer, P. (2012). Quantitative Questions on Attack–Defense Trees. In *ICISC* (Vol. 7839). Springer.

Kordy, B., Piètre-Cambacédès, L., & Schweitzer, P. (2013). DAG-Based Attack and Defense Modeling: Don't Miss the Forest for the Attack Trees. *CoRR*, *1303.7397*. (Available at http://arxiv.org/abs/1303.7397, under submission.)

Lazarus, E. L., Dill, D. L., Epstein, J., & Hall, J. L. (2011). Applying a Reusable Election Threat Model at the County Level. In *Proceedings of the 2011 conference on electronic voting technology/workshop on trustworthy elections* (p. 1-14). Berkeley, CA, USA: USENIX Association.

Legriel, J., Guernic, C. L., Cotton, S., & Maler, O. (2010). Approximating the pareto front of multi-criteria optimization problems. In *Tacas* (p. 69-83).

LeMay, E., Ford, M. D., Keefe, K., Sanders, W. H., & Muehrcke, C. (2011). Model-based security metrics using adversary view security evaluation (ADVISE). In *Eighth international conference on quantitative evaluation of systems, QEST 2011, aachen, germany, 5-8 septemter, 2011* (pp. 191–200). Retrieved from http://dx.doi.org/10.1109/QEST.2011.34 doi: 10.1109/QEST.2011.34

Lenin, A., Willemson, J., & Charnamord, A. (2015, July). Genetic approximations for the failure-free security games. In *Tba.* Springer International Publishing. (undergoing review)

Lin, X., Zavarsky, P., Ruhl, R., & Lindskog, D. (2009). Threat Modeling for CSRF Attacks. In *International Conference on Computational Science and Engineering (CSE'09)* (Vol. 3, pp. 486–491). doi: 10.1109/CSE.2009.372

Lynch, N., & Tuttle, M. (1988). An introduction to input/output automata.

Mauw, S., & Oostdijk, M. (2006). Foundations of Attack Trees. In D. Won & S. Kim (Eds.), *ICISC* (Vol. 3935, pp. 186–198). Springer. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.97.1056

Ou, Y., & Dugan, J. B. (2000, March). Sensitivity analysis of modules dynamic fault trees. In *Computer performance and dependability symposium, 2000. ipds 2000. proceedings. ieee international* (pp. 35–43).

Pièrre-Cambacédès, L., & Bouissou, M. (2010). Attack and defense modeling with BDMP. In *Proc. of the 5th int. conf. on mathematical methods, models and architectures for computer network security (mmm-acns)* (Vol. 6258, pp. 86–101). Springer Berlin Heidelberg.

Reddy, K., Venter, H. S., Olivier, M., & Currie, I. (2008, October). Towards Privacy Taxonomy-Based Attack Tree Analysis for the Protection of Consumer Information Privacy. In *Proceedings of the 6th annual conference on Privacy, Security and Trust (PST '08)* (pp. 56–64). New Brunswick, Canada.

Schneier, B. (1999). Attack Trees: Modeling Security Threats. *Dr. Dobb's Journal of Software Tools*, *24*(12), 21–29. Retrieved from http://www.ddj.com/security/184414879

Tanu, E., & Arreymbi, J. (2010). An examination of the security implications of the supervisory control and data acquisition (SCADA) system in a mobile networked environment: An augmented vulnerability tree approach. In *Proceedings of Advances in Computing and Technology, (AC&T) The School of Computing and Technology 5th Annual Conference* (pp. 228–242). University of East London, School of Computing, Information Technology and Engineering. Retrieved from http://hdl.handle.net/10552/994

Ten, C.-W., Liu, C.-C., & Manimaran, G. (2007). Vulnerability Assessment of Cybersecurity for SCADA Systems Using Attack Trees. In *Proceedings of the IEEE power engineering society general meeting* (pp. 1–8). Tampa, USA.

The TREsPASS Project. (2015). Videos for Analysis Tools. https://vimeo.com/channels/trespass/ (Accessed: October 2015).

The TRESPASS Project, D2.3.2. (2015). *TRESPASS social data and policy extraction techniques.* (Deliverable D2.3.2)

The TRESPASS Project, D3.1.1. (2013). *Initial requirements for quantitative analysis tools.* (Deliverable D3.1.1)

The TRESPASS Project, D3.2.1. (2015). *TRESPASS extraction methods for stochastic models.* (Deliverable D3.2.1)

The TRESPASS Project, D3.3.1. (2013). *First report on stochastic analysis methods.* (Deliverable D3.3.1)

The TRESPASS Project, D5.2.1. (2014). *Currently established risk-assessment methods.* (Deliverable D5.2.1)

The TRESPASS Project, D6.2.2. (2015). *Final refinement of functional requirements.* (Deliverable D6.2.2)

Tidwell, T., Larson, R., Fitch, K., & Hale, J. (2001). Modeling Internet Attacks. In *Proceedings of the 2nd IEEE Systems, Man and Cybernetics Information Assurance Workshop (IAW '01)* (pp. 54–59). West Point, USA.

Willemson, J., & Jürgenson, A. (2010). Serial Model for Attack Tree Computations. In D. Lee & S. Hong (Eds.), *ICISC* (Vol. 5984, pp. 118–128). Springer. Retrieved from http://research.cyber.ee/~jan/publ/serialattack.pdf

A. Project Summary

This chapter gives an overview of the TRE$_S$PASS project and its use cases. The section is shared by the public deliverables to provide the necessary background and to put the current deliverable in context.

Information security threats to organisations have changed completely over the last decade, due to the complexity and dynamic nature of infrastructures and attacks. Attacks like StuxNet involve technical and human factors, and they damage physical infrastructure. The recent attack on a German steel mill [1] was a combination of both targeted phishing emails and social engineering attacks. The phishing helped the hackers extract information they used to gain access to the plant's office network and then its production systems. As a result, the technical infrastructure of the mill suffered severe damage.

The attack on the German steel mill illustrates that we need to integrate the social and technical aspects of systems in assessing their security - and we need to do so today. Socio-technical systems pose new challenges by combining parts for which we often understand the security issues; the combined system is however much more complex due to interactions between these parts.

The main innovation of the TRE$_S$PASS project is the attack navigator, a tool and metaphor that enables defenders to predict and preventing attacks on socio-technical systems. The attack navigator supports current risk-assessment techniques with the TRE$_S$PASS process (developed in Work Package WP5), an analytical approach to identifying attacks and evaluating their impact.

The four main stages in the TRE$_S$PASS process are *data collection*, *modelling*, *analysis*, and *visualisation*. Data collection (WP2) is vital to understanding the nature of a scenario and providing input to subsequent tasks of modelling, analysis and visualisation. Within the project, the focus has been on collection and analysis of social, technical and physical data and the ways in which these relate to one another. Within each of these domains, different approaches have been taken to provide different viewpoints on the nature of the organisation being investigated.

The models (WP1) developed in TRE$_S$PASS can be adapted to the application scenario. We have developed physical modelling techniques in order to understand where further investigation may usefully be targeted. The TRE$_S$PASS model describes relevant aspects of the organisation and their connections. To explore contractual and commercial relationships, the e3value method has been adopted.

[1] BBC News, *Hack attack causes 'massive damage" at steel works*, http://www.bbc.com/news/technology-30575104, last visited October 31, 2015.

The analysis methods (WP3) developed in TRESPASS identify attacks in models and identify the most effective controls to prohibit these attacks. The analyses are supported by tools and together they provide the defender with a comprehensive understanding of properties attacks, *e.g.*, cost for the attacker, required skills, or required time.

The innovative visualisations (WP4) developed in TRESPASS focus focus particularly on visualising elements of the analysis, as this is key to the overall project goal of providing "decision support" to practitioners. However, visualisations contribute also to model development and data gathering.

Practitioners can access the TREsPASS toolkit via the attack navigator map interface, which provides an intuitive means of selecting appropriate tools (WP6) for data gathering, modelling, analysis and visualisation. These can be used, individually or in combination, to strengthen operational and strategic decision-making.

A.1. Case Studies

The TREsPASS process and tools are validated by means of case studies (WP7) in the area of cloud infrastructure, telecommunications infrastructure, ATM infrastructure, and an organisation processing privacy sensitive data.

A cloud infrastructure shares infrastructure within or across organisations, giving the cloud services provider and its employees full physical and logical access to all resources across the different consumers. In TRESPASS we formalise typical components in cloud infrastructures as well as human actors and their interrelationships, to identify their contribution to attacks on the organisation.

In telco infrastructure new products need to be launched under significant time pressure, often opening up loopholes for so-called knowledge insiders who know the market very well, trying to make as much monetary gain from the new products as possible. In TRESPASS we model both the infrastructure and contractual relationships to identify physical and monetary attacks.

The ATM infrastructure connects machines that are composed of a money safe and a computer that controls the ATM's devices. There are well protected ATMs installed inside bank branches, while others are deployed in the street and some are not even embedded in a wall. ATM attacks are common and include classic physical attacks and emerging digital attacks. In TRESPASS we model ATM installations, and identify attack likelihoods using geospatial data.

The organisation processing privacy sensitive data develops a system supporting primarily elderly and disabled people in performing online payments and managing their own money from their home. This case study involves from strictly technical security aspects, such as how information is protected while stored or transmitted, to socio-technical security aspects covering security issues arising from the use of and interaction with the technology. In TRESPASS we identify social-engineering and trust-based attacks on such systems.

A.2. Overview of TRESPASS Integration

The TRESPASS workflow involves several stages with various activities, some of which are optional. Figure A.2 shows the architecture diagram and Figure A.1 shows a visual description of the notation used. In practice, stages may not follow a linear order. For example, depending on the goal of the risk assessment, new data requests may be issued later in the process, or automatic updates of data may be supported.

The **Data collection stage** prepares for analysis and modelling steps, and may require the gathering of one or more of the following kinds of data.

Physical data collection provides knowledge about the physical layout of the organization including locations, buildings, rooms, doors, windows, etc.

Digital data collection gathers information about the organization's IT infrastructure.

Social data collection focuses on organisational and individual data, and results in actor profiles containing, *e.g.*, attributes of employees, stakeholders, or potential attackers.

Commercial data collection gathers information required for *e3fraud* analyses, which focus on potential fraud.

Stakeholder goal collection identifies assets and policies the protection of which is critical to one or more stakeholders.

The **model creation stage** handles the creation of the TRESPASS model and associated actor profiles. The e3value model creation process is complementary to the main TRESPASS model, for cases requiring a more specific financial focus:

TRESPASS model creation is a key activity result in a system model that can be further extended and analysed.

- **Components customization (optional)** takes place before or during the TRESPASS model creation to create specialized custom model components.
- **Attacker profile creation** creates the attacker profile that the TRESPASS model analysis should consider, based on ready-made attacker profiles.
- **Defender/target profile creation** creates similar profiles for the other actors in the model based on the social data gathered in the social data collection activity.

e3value model creation This interactive activity involves using the *e3value toolkit* [2] to create business value models. These models structure the commercial information gathered in the data collection stage in a formal way.

In the **analysis stage** different analyses are possible depending on the model chosen. The analysis of the TRESPASS model involves these steps:

1. In the **attacker profile selection**, the user selects the attacker profile to use in the analysis.

[2] http://e3value.few.vu.nl/tools/

2. The **attacker goals creation** provides the attack generation with the attacker goals. These can be derived by hand from the stakeholder goals or deduced automatically from the selected attacker profiles.
3. The **scenario selection** selects a scenario, consisting of a single pair of attacker and attacker goal, to run the TRESPASS analysis on.
4. To extend attack trees, **attack pattern creation and sharing** provides libraries with known attack steps. The attack tree generation can only reach a certain level of abstraction, which may not be sufficient for quantitative analyses.
5. **Attack generation** transforms the TRESPASS model to an attack tree.
6. **Attack tree annotation & augmentation** then extends the attack tree with attack patterns and decorates leaf nodes with parameter values from the data collection stage for quantitative analysis.
7. The **attack tree analyses** compute quantitative properties of attacks, *e.g.*, utility for the attacker or success probability of the attack.

The analysis of the **e3value model** is complementary to the core TRESPASS analysis and has only one step:

1. For the **fraud model generation**, the user needs select an attacker and an interval of expected occurrence rates of the commercial transactions specified by the e3value model. The e3fraud tool then identifies all possible violations of contracts, the loss for actors, and the delta in profit for the other actors.

The **visualisation stage** can be used continuously to provide practitioners with feedback regarding the results of their activities:

1. **Fraud model visualisation** shows the generated attacks as a ranked list of textual descriptions of the attack steps and displays charts showing the profitability for each actor.
2. **Attack tree visualisation** shows the intermediary attack trees.
3. **Attack tree analysis visualisation** visualises analysis results.

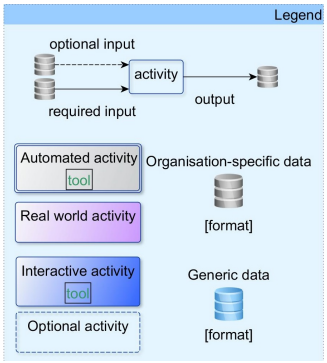

Figure A.1.: Legend for the Integration diagram in Figure A.2.

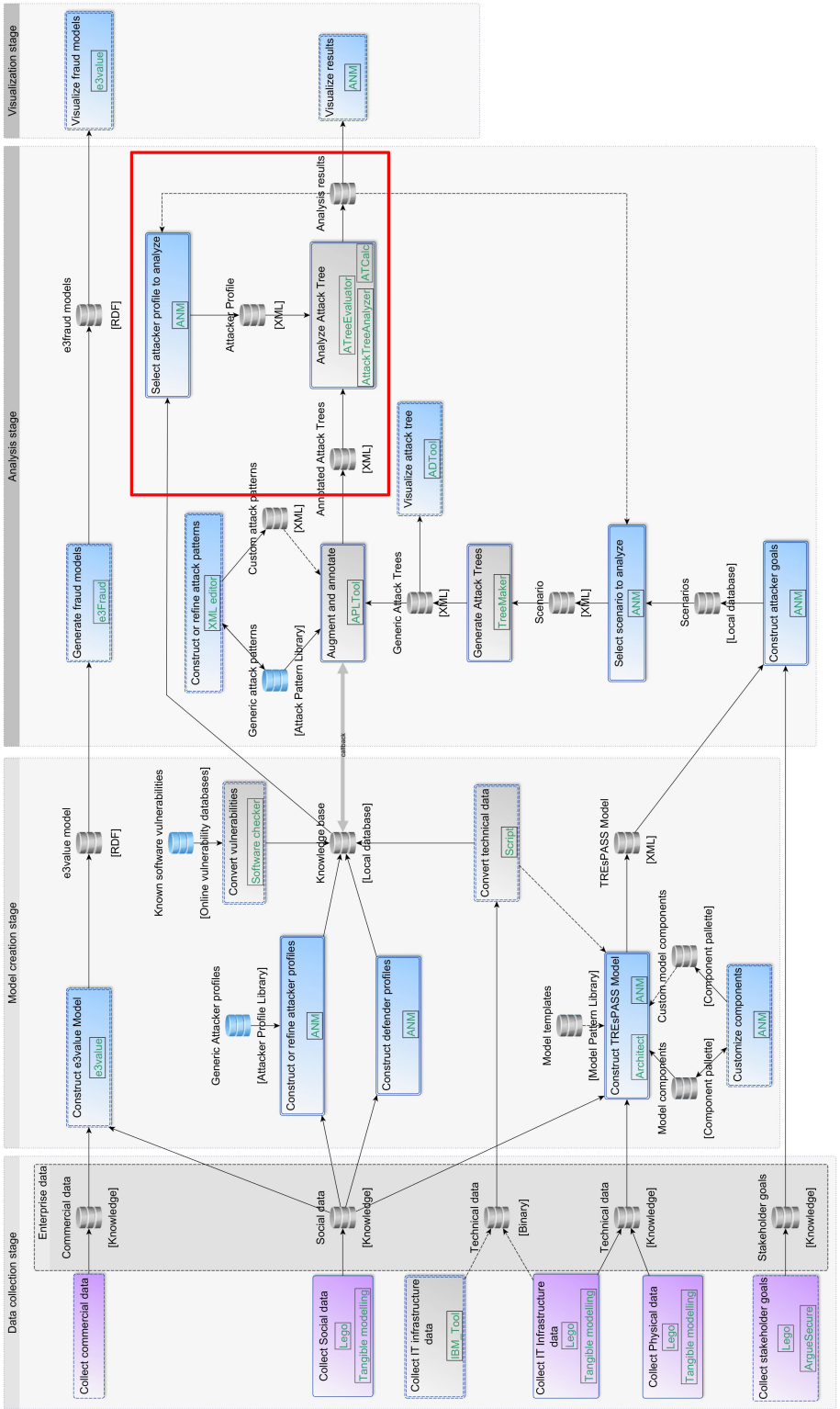

Figure A.2.: Integration diagram for the TRESPASS project.

B. Analysis Result of the Estonian i-Voting Attack Tree

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<adtree utility="989784.62">
<node refinement="disjunctive">
<label>Voting tree</label>
<node refinement="disjunctive">
<label>Revocation attack</label>
<node refinement="disjunctive">
<label>Attack integrity</label>
<node refinement="conjunctive">
<label>Attack voter</label>
<node refinement="disjunctive">
<label>Convince NEC RL19</label>
<parameter class="numeric" name="cost">3000.00</parameter>
<parameter class="ordinal" name="difficulty">M</parameter>
<parameter class="ordinal" name="likelihood">H</parameter>
<parameter class="ordinal" name="time">HR</parameter>
</node>
<node refinement="disjunctive">
<label>Compromise voters computers</label>
<node refinement="conjunctive">
<label>Fake voting applications</label>
<node refinement="disjunctive">
<label>Distribute fake Voting App</label>
<node refinement="conjunctive">
<label>Use fake website</label>
<node refinement="disjunctive">
<label>Develop fake website RL12</label>
<parameter class="numeric" name="cost">800.00</parameter>
<parameter class="ordinal" name="difficulty">M</parameter>
<parameter class="ordinal" name="likelihood">V</parameter>
<parameter class="ordinal" name="time">HR</parameter>
</node>
<node refinement="disjunctive">
<label>Get voters to visit fake website</label>
<node refinement="disjunctive">
<label>Social media RL15</label>
```

```
<parameter class="numeric" name="cost">400.00</parameter>
<parameter class="ordinal" name="difficulty">L</parameter>
<parameter class="ordinal" name="likelihood">V</parameter>
<parameter class="ordinal" name="time">HR</parameter>
</node>
</node>
</node>
</node>
<node refinement="disjunctive">
<label>Develop fake Voting App RL11</label>
<parameter class="numeric" name="cost">3840.00</parameter>
<parameter class="ordinal" name="difficulty">M</parameter>
<parameter class="ordinal" name="likelihood">V</parameter>
<parameter class="ordinal" name="time">HR</parameter>
</node>
</node>
</node>
</node>
</node>
</node>
</node>
</adtree>
```