predict
prioritise
prevent

TRESPASS

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

Deliverable D3.4.2

Methods for Attack Generation, Preventive Measures, and Ranking

Project:	TRE$_S$PASS
Project Number:	ICT-318003
Deliverable:	D3.4.2
Title:	Methods for Attack Generation, Preventive Measures, and Ranking
Version:	1.0
Confidentiality:	Public
Editor:	Rolando Trujillo
Cont. Authors:	R. Trujillo, Z. Aslanyan, O. Gadyatskaya, R. R. Hansen, A. Lenin, G. Lenzini, R. Kumar, C. W. Probst, M. Ford, C. Muller, D. Ionita
Date:	2016-10-31

SEVENTH FRAMEWORK PROGRAMME

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
UL	O. Gadyatskaya	All
	R. Trujillo	All
CYB	A. Lenin	7
DTU	Z. Aslanyan	7
	C. W. Probst	3, 7
AAU	R. R. Hansen	7
ITR	C. Muller	1, 2, 5
UT	M. Stoelinga	5
	D. Ionita	6
	R. Kumar	7
CHYP	M. Ford	All

Quality assurance		
Role	Name	Date
Editor	Rolando Trujillo (UL)	2016-10-31
Reviewer	Henk Jonkers	2016-10-15
Reviewer	Dan Ionita	2016-10-15
Task leader	Jan Willemson (CYB)	2016-10-31
WP leader	Jaco van de Pol (UT)	2016-10-31
Coordinator	Pieter Hartel (UT)	2016-10-31

Circulation	
Recipient	Date of submission
Project Partners	2016-09-30
European Commission	2016-10-31

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 (TRESPASS). 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

This deliverable reports on recent advances on attack generation, preventive measures, and ranking. The report, in essence, allows the reader to understand the analysis process from the perspective of the end-user of TRE$_S$PASS. The deliverable also contains a portfolio of the analysis tools developed by WP3, which were produced jointly in tasks T3.2 – T3.5. This part of the deliverable allows the reader to zoom into our analysis methods and to understand better how the analysis processes run under the hood.

Key takeaways:

- The TRE$_S$PASS project provides three novel methods for attack generation (Chapter 3). Two of them rely on socio-technical security models (Sections 3.1 and 3.2), while the third one is a systematic approach for the manual generation of attack scenarios (Section 3.3).
- The approaches for attack generation have been extended in such a way that preventive measures can be added to system and attack models (Chapter 5). Preventive measures are typically captured in the form attack-defence tree models or attack-defence diagrams (Chapter 2).
- Because financial fraud is a type of attack with its own peculiarities, the project has developed a dedicated method and tool for fraud assessment (Chapter 6), based on the e^3*value* modelling language.
- The TRE$_S$PASS tools, documented in Chapter 7, implement stochastic analysis and perform ranking of attack scenarios (Chapter 4). Many of these tools have been already documented in (The TRE$_S$PASS Project, D3.3.2, 2015). Others are revamped tools, e.g. the ADTool 2.0 (Section 7.2), or completely new tools (e.g. AT-Top (Section 7.4), which have been only recently added to the TRE$_S$PASS analysis toolkit.

1 Introduction

1.1 Goals

In the following sections we provide a synthesis of the main goals of WP3 that will allow the reader to understand better the setting and the results achieved in Task T3.4.

1.1.1 WP3 goals

WP3 focuses on quantitative analysis tools that can be used to measure security-related attributes based on socio-technical models of organizations. Figure 1.1 outlines the different tasks of WP3 and its most important links to other work packages. More details about integration of work packages are given in deliverables D3.1.1 (The TRESPASS Project, D3.1.1, 2013) and D3.1.2 (The TRESPASS Project, D3.1.2, 2015). Figure 1.1 shows that WP3 is tightly coupled with WP1, as our analysis techniques work based on a socio-technical security model. In WP1 the socio-technical security model is created based on the data collected by WP2. The analysis results are visualised by WP4 (visualisation process and tools).

As reported in previous deliverables (D3.1.1 (The TRESPASS Project, D3.1.1, 2013), D3.3.1 (The TRESPASS Project, D3.3.1, 2013), D3.4.1 (The TRESPASS Project, D3.4.1, 2014), D3.1.2 (The TRESPASS Project, D3.1.2, 2015), D3.2.1 (The TRESPASS Project, D3.2.1, 2015) and D3.3.2 (The TRESPASS Project, D3.3.2, 2015)), WP3 comprises the following main tasks:

- Task T3.1 designs and documents a set of functional requirements for the analysis tools.
- Task T3.2 extracts stochastic models from domain-specific attack models (such as attack trees).
- Task T3.3 develops analysis methods for the extracted low-level stochastic models.
- Task T3.4 generates domain-specific attack models from the socio-technical model and analyses these models. It also ranks the found attacks based on the analysis results (these can be results received as output from task T3.3 or obtained directly in task T3.4 by running analyses on the domain-specific attack models), and advises countermeasures based on top-ranked attack scenarios.
- Task T3.5 investigates dynamic aspects of stochastic models that can improve the analysis results.

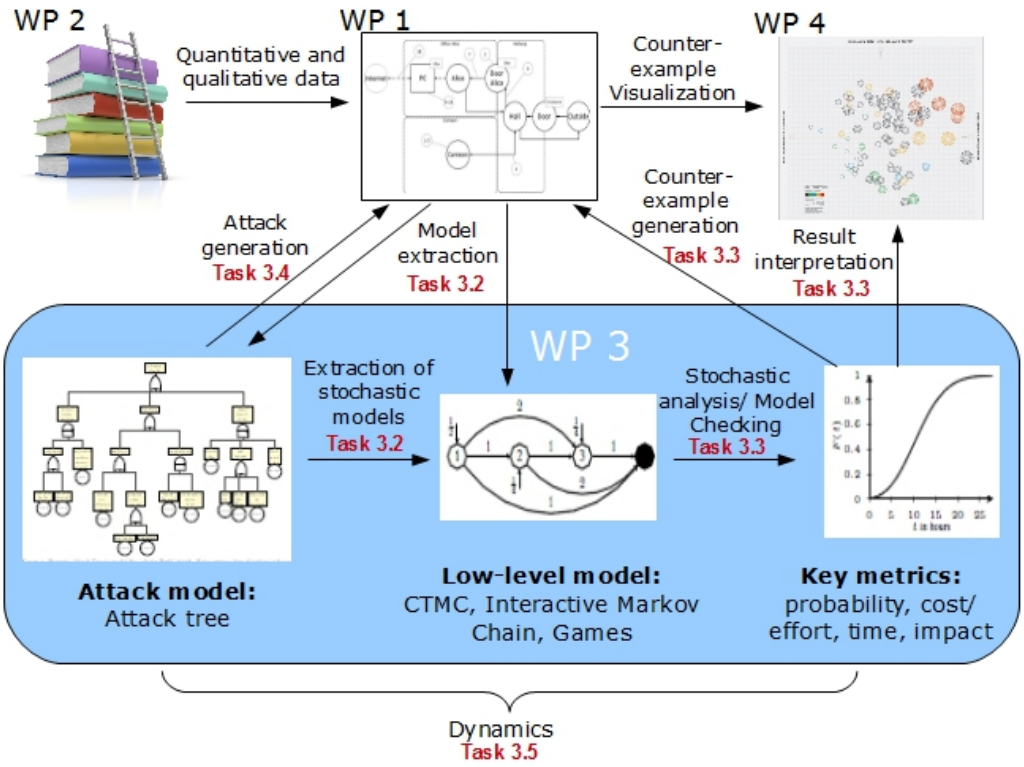


Figure 1.1: The tasks of WP3 and links with WP1, WP2 and WP4 within TRE$_S$PASS. The blue shaded area contains the architecture of WP3 in terms of used models. Arrows between two models or between a model and another WP indicate their relationship: An arrow between A and B indicates that A provides inputs to B, or B is dependent on A.

1.1.2 Goals of this deliverable

This deliverable builds upon the work on the restricted deliverable (The TRE$_S$PASS Project, D4.3.1, 2014), and its main goals are twofold. First, consolidating the findings by the TRE$_S$PASS consortium on attack generation, preventive measures, and ranking. Attack generation has been partially addressed in D1.3.4 (The TRE$_S$PASS Project, D1.3.4, 2016), while ranking is implicitly considered in most analysis methods developed within the TRE$_S$PASS project (The TRE$_S$PASS Project, D3.3.2, 2015). The second goal of this deliverable is to describe the portfolio of analysis tools developed in TRE$_S$PASS.

1.2 Document structure

For the sake of completeness, Chapter 2 reviews several attack and system models frequently used in TRE$_S$PASS. Details on those models can be found in other deliverables, e.g. (The TRE$_S$PASS Project, D1.3.4, 2016). Chapters 3 and 5 show, respectively, the

latest progress on the generation of attacks and countermeasures. Ranking of found attacks is described in Chapter 4. A dedicated model and methodology to analyse fraud scenarios is introduced in Chapter 6. Finally, in Chapter 7 we present the details of the analysis tools and methods developed by the TRESPASS project in WP3. We close the document with concluding remarks in Chapter 8.

1.3 Foreground and background

There exists a rich body of knowledge concerning attack trees, attack-defence trees, timed automata, socio-technical models, and other underlying techniques and models that we use as a background in our approach. Furthermore, the ADTool has existed before the project and has served as a background to it. The current version of the ADTool was extended in TRESPASS, as well as the formalization of Sequential Attack trees. All other reported achievements and tools are foreground of the project, which represents roughly 85% of this document.

2 Attack and system models

Many methodologies exist for risk assessment activities, but graphical security models help to illustrate and guide the consideration of security throughout the lifecycle of a product, system or organisation. They are very powerful in the elucidation of attacks and countermeasures thanks to their visual properties (Schweitzer, 2013).

A number of attack and system models have been explored in the project, in order to identify the comparative strengths of each as potential components of the final TRE$_S$-PASS processes. Certainly one should not look at them as separate components, but as a pragmatic and flexible process allowing the analyst to conveniently move from one model to another. An example of such a journey is depicted in Figure 2.1, where the starting point is a complex socio-technical model that can be transformed into more fine-grained attack models, such as timed automata or attack trees. Analysis results on those models can be used to feed and update the initial socio-technical model. We thus detail next relevant attack and system models, which are used in TRE$_S$PASS for attack generation, preventive countermeasures, and ranking.

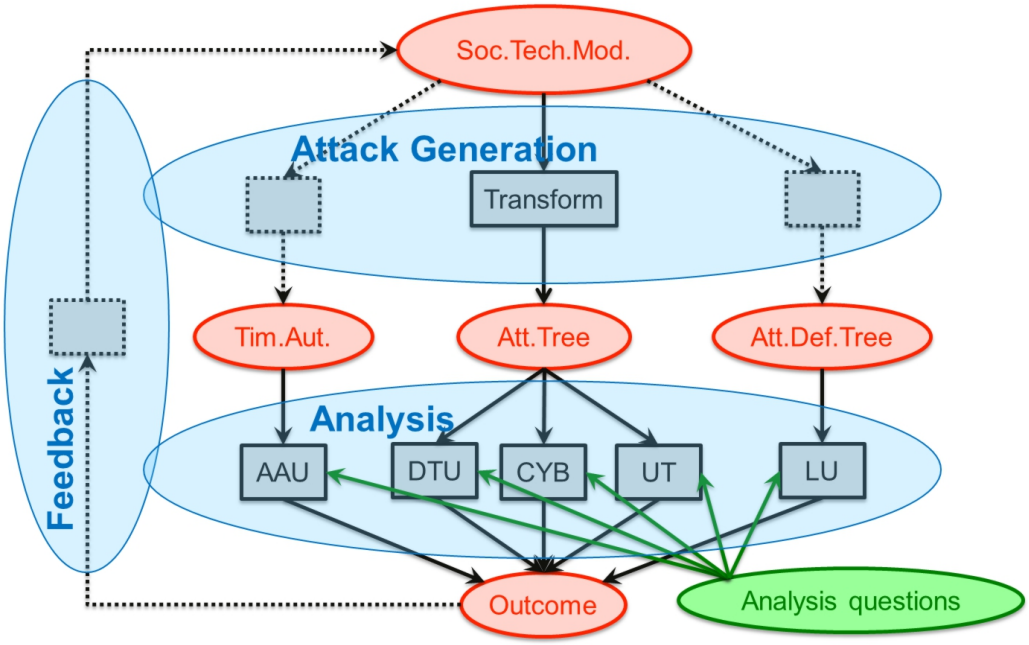


Figure 2.1: Interaction between system and attack models.

2.1 Attack trees

In order to better assess the security level of a complex and heterogeneous system, a gradual refinement method called a threat tree or attack tree method can be used. The basic idea of the approach is simple – the analysis begins by identifying one or more of the goals of the primary attacker and continues by splitting these into subgoals (sub-attacks), with either all or some of them being necessary to materialise the primary goal. Sub-attacks can be refined further, until it is no longer necessary to refine the resulting attacks any further; non-refined attacks are typically called *elementary* or *atomic attacks*. The result of this refinement process is an attack model with a tree structure, where the root node represents the primary attacker's goal and the leaf nodes the elementary attacks.

Similar models to attack trees have been extensively used for decades. For example, fault assessment of critical systems is commonly addressed by means of fault trees (Vesely, Goldberg, Roberts, & Haasl, 1981), while software vulnerability analysis considers a similar model called threat tree (Viega & McGraw, 2001; Moore, Ellison, & Linger, 2001). Indeed, the term attack tree started to be widely adopted by the information security community after Bruce Schneier work (Schneier, 1999).

Attack trees can also be used for sound quantitative analysis (Mauw & Oostdijk, 2005). In order to do so, every leaf node in the tree should be annotated with attribute values. Attribute values can be of different types, e.g. boolean values to represent feasibility or numerical values to represent cost or probability of occurrence of an attack.

2.2 Attack trees with sequential conjunction

Because the attack tree methodology was not intended to model dynamic systems (e.g. attacker/defender interaction, dependent actions, or random failures), it has been naturally extended to capture such dynamics. For example, (Bistarelli, Fioravanti, & Peretti, 2006) proposed an approach to extend the leaf nodes of an attack tree with a set of countermeasures. This notion of integrating the defender's behavior into an attack tree was significantly enhanced by (B. Kordy, Mauw, Radomirović, & Schweitzer, 2012a) as *attack–defence trees* and (Roy, Kim, & Trivedi, 2012a) as *attack countermeasure trees*.

Sequential operators can also enhance attack trees by modelling a sort of dynamic behaviour (e.g. dependency) while keeping the simplicity and scalability of static modelling. In 2005, Wen-ping and Wei-min incorporated the use of the SAND operator in their *improved attack trees*, to model attacks on information transmission links and acquisition systems (Lv & Li, 2011). (Piètre-Cambacédès & Bouissou, 2010) propose a modelling approach combining attack trees and Markov chains to encode dynamic security processes. (Camtepe & Yener, 2007) consider attacks with expiration time and temporal dependencies. (Jürgenson & Willemson, 2009) also introduced a temporal order for elementary attacks, their approach was further developed in (Jürgenson, 2010) and (Niitsoo, 2010).

Within the TRESPASS project we have also experienced the need for defining and modelling actions as a sequence rather than as a set of steps. In particular, the attack tree model designed for the IPTV case study (The TRESPASS Project, D7.1.1, 2013) contains a plethora of refinements that can only be interpreted as a sequential composition. Another example showing the need for incorporating sequential conjunction in attack trees is the attack tree generation process explained in Chapter 3.

For these reasons, in TRESPASS we have proposed a rigorous mathematical formalisation of the sequential AND operator in the context of attack trees, called SAND attack trees (Jhawar, Kordy, Mauw, Radomirovic, & Trujillo-Rasua, 2015). The formalisms resembles other foundational papers that propose formal semantics for attack trees (Mauw & Oostdijk, 2005) and their extensions (B. Kordy et al., 2012a). The proposed semantics (SP semantics) is based on series-parallel (SP) graphs, which are edge-labeled directed graphs with a single source vertex and a single terminal vertex. A complete axiomatization for the SP semantics shows that the SP semantics of SAND attack trees are a conservative extension of the multiset semantics of standard attack trees (Mauw & Oostdijk, 2005) (i.e. the extension does not introduce unexpected equivalences w.r.t. the multiset semantics). We have also proved that the domains of SAND attack trees and sets of SP graphs are isomorphic. Finally, by introducing the notion of attributes for SAND attack trees, we enable quantitative analysis of attack scenarios using the standard bottom-up evaluation algorithm.

2.3 Attack-defence trees

In 2010, attack countermeasure trees (ACTs) (Roy, Kim, & Trivedi, 2010a, 2010b) were proposed as a means of modelling both attacking and defensive strategies. The methodology was further developed in (Roy et al., 2012a) by considering three distinct classes of events: attack events, detection events and mitigation events. The classical AND and OR nodes, as defined for attack trees, were also extended in ACTs; both were generalised by introducing a new k-out-of-n node.

In parallel to ACTs, attack–defence trees (ADTrees) were proposed by (B. Kordy, Mauw, Radomirović, & Schweitzer, 2010) in 2010. The main difference between the two models is that the latter can model interleaving attacker and defender actions. Indeed, a strong connection between game theory and graphical security assessment using ADTrees has been proven to exist (B. Kordy, Mauw, Melissen, & Schweitzer, 2010).

The intuitive graphical nature of ADTrees enables them to bridge the gap between stakeholders from diverse backgrounds, providing an environment to brainstorm, amend, document and analyse a wide range of threats. In particular, ADTrees provide a succinct and meaningful structure for a huge number of potential attack vectors. This strength was already prominent when Schneier introduced attack trees back in 1999 (Schneier, 1999). However, attack trees are constructed from the attacker's perspective. Organisations are

more focused on the overall risk (in terms of worst case impact) and the inventory of effective treatment options to be implemented to mitigate those risks. This is where ADTrees are more useful than attack trees.

The advantages of ADTrees are that they allow the analysts (security experts or risk assessors) to link countermeasures to the original attacks of the ATree they apply to, thus helping to focus on interactions between attackers and defenders. Indeed, the ADTree model can be seen as a game between two players, the proponent and the opponent. When the root of the tree is an attack node, the proponent is an attacker and the opponent is a defender, and the opposite when the root is a defence node. The children of the root in an ADTree are refinements of the global goal of the proponent.

ADTrees have been enriched with several semantics, e.g. propositional semantics and semantics induced by a De Morgan lattice (B. Kordy, Mauw, Radomirović, & Schweitzer, 2010). A semantics is typically interpreted as an equivalence relation over the universe of ADTrees, which allows for the definition of equivalent ADTree representations of a scenario. These semantics, together with an attribute domain, facilitate the extension of the standard bottom-up algorithm, formalized for attack trees in (Mauw & Oostdijk, 2005), to ADTrees in (B. Kordy, Mauw, Radomirović, & Schweitzer, 2010). Consequently, in addition to extending the modelling capabilities of attack trees, ADTrees do not increase the computational complexity of the model.

The potential of the ADTree methodology has been demonstrated in an extensive case study of a real-life RFID goods management system (Bagnato, Kordy, Meland, & Schweitzer, 2012). The study was performed by academic and industrial researchers from different backgrounds.

2.4 Attack-defence diagrams

In TRE$_S$PASS we have extended the attack-defence tree formalism in order to represent intricate interaction between attacker and defender (Hermanns, Krämer, Krcál, & Stoelinga, 2016). The extension is named *Attack Defence Diagram (ADD)*. ADDs are akin to, but significantly more expressive than attack trees and their variants like attack-defence trees (B. Kordy, Mauw, Radomirović, & Schweitzer, 2011), attack countermeasure trees (Roy, Kim, & Trivedi, 2012b), and attack graphs (Ingols, Chu, Lippmann, Webster, & Boyer, 2009). ADDs are acyclic graphs that express how basic steps can be combined into a game between attacker and defender, competing to swing the game from being undecided (uu) over to either a successful attack, or a successful defence of the system. This genuine game perspective is reflected semantically by the use of three-valued logic (3VL) to characterise the dynamic evolution of the game. In this, the local context determines whether 'true' or 'false' (tt or ff) corresponds to success or defeat of attacker, or dually defender.

The game outcome may crucially depend on aspects of time, probability, and cost, reflecting the timing of attack steps and countermeasures, their success chances, as well

as skills, knowledge and budget of attacker and defender. This links to ongoing and substantial (Mateski et al., 2012) activities to develop *cyber threat metrics* that are aimed at providing characteristics of (attacker) moves in terms of cost, time, detection probability or success probability.

To represent these aspects, basic events in ADD are equipped with relevant quantitative information about duration, success probability, and cost. ADD gates then express the dynamics of how attacks and defences interact and propagate through the system, gradually determining the game. Apart from standard gates like `AND`, `OR`, and their sequential versions, we introduce several new gates and an assembly box for other gates facilitating security modelling. Also cyclic or repetitive behaviour is supported by ADD.

From the expressiveness perspective, ADDs unite and over-arch ADVISE and attack defence trees and graphs. ADVISE (LeMay, Ford, Keefe, Sanders, & Muehrcke, 2011) is a powerful security analysis framework with analysis capabilities for a wide number of quantitative security metrics. However, ADVISE provides limited syntactic constructs for security modelling. Attack trees and attack graphs do provide such syntactic aids, as well as quantitative analysis methods. However, to the best of our knowledge, none of the approaches thus far provides a comprehensive framework combining cost, probability, time, defences, and choices of players in a game interpretation. Also, we believe that current versions of attack defence trees and graphs lack expressivity when in comes to modelling real defence measures.

The dynamic quantitative interpretation of ADDs is realized via the Modest framework (Hahn, Hartmanns, H., & Katoen, 2013), a state-of-the-art stochastic modelling formalism with powerful analysis capabilities and tool support. In the style of (Kumar, Ruijters, & Stoelinga, 2015a) we provide a compositional translation from ADD to Modest. That is, we translate each ADD modelling construct into a Modest process and then compose these processes to obtain the entire game model. In this way, we can analyse various security metrics for the ADD model, for example, we can *perform what-if analysis* for fixed strategies for both players and *investigate game-related questions* like best responses to player actions. Further below, in Section 5.3, we provide an intuitive discussion on how ADDs can be used to derive security metrics.

2.5 Timed automata

The *timed automata* formalism, first defined in Alur and Dill (1994), was designed as the formal semantic foundations for models of distributed systems with real-time clocks and clock-constraints. A timed automaton can be seen as an extension of a finite automaton with real-valued clocks and the possibility of restricting transitions based on a value computed over these clocks. Numerous extensions and variants of the basic timed automaton have been explored and investigated. With a focus on modelling systems with cost and stochastic behaviour respectively, especially the so-called *priced timed automata* and *probabilistic timed automata* are of particular interest, in addition to the basic timed automata in the TRE$_S$PASS project.

One of the primary advantages of modelling and analysis with formalisms based on timed automata, is that there is extensive and mature tool support for many timed automata variants and for many different purposes, including modelling, analysis, and verification. In the TRESPASS project, the UPPAAL[1] model checker is the primary model checking tool used (Behrmann, David, & Larsen, 2004). Many of the extended/modified timed automata formalisms also have good tool support, e.g. UPPAAL TIGA for model checking of safety and reachability properties in *timed games*, UPPAAL SMC for *statistical model checking* (David, Larsen, Legay, Mikucionis, & Wang, 2011) which supports statistical model checking of hybrid models.

The combination of expressiveness and powerful tool support, has made timed automata and model checking widely used in many modelling and verification projects, ranging from academic research projects to real-life industrial cases.

2.6 Socio-Technical security models

Socio-Technical Security (STS) models view a physical system as a physical space (e.g., a building) containing objects and actors. The space is structured in locations (e.g., rooms) accessible via lockable doors. People act in such a space by performing actions within a predefined given set. For instance, they lock or unlock doors, enter and exit rooms, manipulate and exchange objects. Each action may happen non-deterministically, conditionally, or probabilistically. In the same space, a malicious actor, the intruder, threatens people's normal workflow.

Within the TRESPASS project we have developed a dedicated STS model, the so-called TRESPASS model, which supports automated generation of attack scenarios (The TRESPASS Project, D1.3.4, 2016). The consortium has also extended the TRESPASS model with the aim of computing attribute values, such as probability and cost, directly in the STS model (Lenzini, Mauw, & Ouchani, 2015). We provide details on both models next.

2.6.1 The TRESPASS socio-technical security model

The TRESPASS model is described in detail in (The TRESPASS Project, D1.3.4, 2016). It models the physical, technical, and social layer, and provides the input for Treemaker, which identifies possible attacks on the model (c.f. Sections 3.1 and 7.1).

A TRESPASS model consists of the following components:

- Actors represent the social component.
- Locations are connected by edges and represent the physical infrastructure.
- Items and data represent physical and virtual items and data, and can be located at each other:

[1] http://uppaal.org/

  - Items can be located at other items, at locations, as well as at actors and processes; depending on where they are located they are either physical or virtual items.
  - Data can be located at items, at locations, and also at actors and processes; like items, depending on where data is located, it is either physical or virtual.
- Processes consist of sequences of actions that work on virtual data and items; the modelled actions perform input and output (In, Out), move a process (move), or start a new process at another location (eval).
- Actors can perform the same actions as processes, however we usually do not model their actions; instead they are assumed to perform whatever actions necessary to reach a goal.
- Last but not least, policies describe, which actions may be performed at a certain location, and which credentials must be presented in order to perform the action.

To map model elements to their properties, and to support the mapping of analysis results back into the model, each model component has a unique identifier, which is used in retrieving, updating, and storing information.

2.6.2 Lenzini, Mauw and Ouchani's model

Lenzini, Mauw and Ouchani consider a system to be a tuple $S = \langle Phy, Obj, Act, Struc \rangle$, where Phy stands for *physical space*, Obj for *objects*, Act for *actors*, and $Struc$ for *structure* (Lenzini et al., 2015). The interpretation of these components is the following:

- Physical space: Phy is a building's infrastructure, including its locations (e.g. rooms) and the doors connecting these. It relates keys to the doors that they open and provides information on whether a door is locked or not.
- Objects: Objects can be of various natures, for instance, keys, doors, containers (e.g. a safe), assets, etc. An object can be movable, destroyable, or both. Besides, doors and containers are lockable.
- Actors: Actors represent the persons interacting with objects and among them in the physical space. An actor has a behaviour, modelled as a sequence of basic actions, which can be executed non-deterministically, conditionally, or probabilistically. Possible actions include moving from one location to another, locking or unlocking a door or container, as well as taking, dropping or destroying an object.
  - Intruder: The intruder is a particular actor, which differentiates from the other, honest actors, in the fact that he has fewer restrictions. Firstly, he can execute any action which is possible, in the sense that the infrastructure and the objects allow it. Secondly, he can execute actions that honest actors cannot. For example, he can unlock doors without holding the appropriate key, pick-pocket objects from other actors, "putpocket" objects (i.e. surreptitiously slip objects into an honest actor's pocket), or impersonate anyone in the exchange

of objects with others. However, the intruder cannot move between locations not connected through doors, move unmovable objects, destroy undestroyable objects, pick or drop objects in locations different from the one where he is standing.

- Structure: The structure describes, via a directed graph, the hierarchical relations between locations, doors, objects and actors. It links locations through doors, actors to locations, and objects to locations, other objects (e.g. containers) or actors.

The main differences between this model and the TRE$_S$PASS model is that the former supports analysis through a probabilistic model checker, and it also supports the open and destroy actions. In the TRE$_S$PASS model, these components are provided by the knowledge base (The TRE$_S$PASS Project, D2.4.1, 2016) and the analysis methods (The TRE$_S$PASS Project, D3.3.2, 2015).

3 Attack generation

Developing attack models for complex systems has been traditionally a cumbersome task. Depending on the system, some attack models can be more appropriate than others. For this reason, the TRE$_S$PASS project has investigated different approaches for attack generation, which essentially rely on different attack models. In this section we introduce our main approaches to automated attack generation. Finally, we introduce a novel process for the manual creation of multi-stage attack scenarios via attack-defence trees (Fraile et al., 2016).

3.1 Automated generation of attack scenarios

The automated generation of attack scenarios takes as input a TRE$_S$PASS socio-technical security model (The TRE$_S$PASS Project, D1.3.4, 2016) and a goal to be invalidated. This goal is expressed as a policy (The TRE$_S$PASS Project, D1.2.2, 2015), which can represent different situations:

- Asset goals represent assets that an attacker should not be able to obtain,
- Action goals represent actions that an attacker should not be able to perform, and
- Location goals represent locations, that an attacker should not be able to reach.

On a high level, our approach aims at invalidating the goal (Ivanova, Probst, Hansen, & Kammüller, 2015c, 2015a) chosen by performing four basic steps:

1. Choose the policy to invalidate, and identify the possible actors who could do so; these are the potential attackers.
2. Depending on the goal, do
   - For asset goals identify the set of locations where the asset can be obtained,
   - For action goals identify the set of locations where the prohibited actions can be performed.
   - For location goals, return the set with the goal location.
3. Recursively generate attacks for fulfilling any of these goals (obtaining an asset, performing an action, or reaching a location). This will also identify required assets to perform any of these actions, and obtain them.

4. Finally, move to the location identified in the second step and perform the action, obtain the asset, or reach the location.

In the following we describe the rules for asset goals; action goals and location goals are handled accordingly, either by assuming the action to be inputting an asset, or the location to be the location where the asset is located.

It should be noted that all rules either block if no valid result can be computed, or return an empty attack tree, for example, if no credentials are required. The rules take as input an infrastructure component $\mathcal{I}$, which represents the TRE$_S$PASS socio-technical security model (The TRE$_S$PASS Project, D1.3.4, 2016), and an actor component $\mathcal{A}$, which stores identities, locations, and assets collected and reached by an actor during an attack. Also note that we extend rules from working on singular elements to sets by unifying the results of rule applications.

Identify Attackers. To start attack generation from a global policy (see Figure 3.1), we compute the unification of the global policy and the set of all actors, identify the set of attackers by means of function $getAttacker$, which replaces a variable with the identified bindings, or returns an explicitly specified attacker:

$$getAttacker_{\mathcal{I}}(a, \sigma) \quad := \quad \begin{cases} \{a\} & \text{if } a \in \mathsf{N}_a \\ \sigma(a) & \text{if } a \in Vars \end{cases}$$

$$\frac{\begin{array}{c} \sigma = unify_{\mathcal{I}}(Actors, credentials) \\ attackers = getAttacker_{\mathcal{I}}(actor, \sigma) \qquad goals = applicableAt_{\mathcal{I}}(\textit{credentials}, \textit{enabled}, \sigma) \\ \mathcal{I}, attackers, goals \vdash_{goal} trees \qquad \mathcal{T} := \oplus_{\vee}^{\text{“perform any actions”}} trees \end{array}}{\mathcal{I}, not(actor, credentials, enabled) \vdash_{P} \mathcal{T}}$$

Figure 3.1: Attack generation starts from the global action-based policy $not(actor, credentials, enabled)$. Attack trees are generated for all possible policy violations. As every attack tree represents a violation of the policy, the resulting attack trees are combined by an *or* node.

$$\frac{\mathcal{I}, \mathcal{A}, goto(location) \wedge perform(action) \vdash_{GP} \mathcal{T}}{\mathcal{I}, \mathcal{A}, (location, action) \vdash_{goal} \mathcal{T}}$$

$$\frac{\mathcal{I}, \mathcal{A}, goto(l) \vdash_{goto} \mathcal{T}_{goto}, \mathcal{A}' \qquad \mathcal{I}, \mathcal{A}', perform(a) \vdash_{perform} \mathcal{T}_{action}, \mathcal{A}''}{\mathcal{I}, \mathcal{A}, goto(l) \wedge perform(a) \vdash_{GP} \mathcal{T}_{goto} \oplus_{\wedge}^{\text{“goto } l \text{ and perform } a\text{”}} \mathcal{T}_{action}, \mathcal{A}''}$$

Figure 3.2: For each identified goal (consisting of a location and an action) an attacker moves to the location and performs the action. The rules result in an attack tree and a new state of the attacker, which includes the obtained keys and reached locations.

Identify Target Locations. We then compute all locations at which one of the actions in $enabled$ could be applied using the credentials specified in the policy. The function

$applicableAt$ identifies all these locations in the system model and returns goals as pairs of actions and locations.

$$\frac{paths = getAllPaths_{\mathcal{I}}(\mathcal{A}, l) \qquad \mathcal{I}, \mathcal{A}, paths \vdash_{path} trees, \mathcal{A}' \qquad \mathcal{T} := \oplus_{\vee}^{\text{“find path to } l\text{”}} trees}{\mathcal{I}, \mathcal{A}, goto(l) \vdash_{goto} \mathcal{T}, \mathcal{A}'}$$

$$\frac{missing = missingCredentials_{\mathcal{I}}(\mathcal{A}, path) \qquad \mathcal{I}, \mathcal{A}, missing \vdash_{credential} trees, \mathcal{A}' \qquad \mathcal{T} := \oplus_{\wedge}^{\text{“get credentials”}} trees}{\mathcal{I}, \mathcal{A}, path \vdash_{path} \mathcal{T} \oplus_{\wedge}^{\text{“get credentials and pass path”}} \mathcal{N}^{\text{pass path}}, \mathcal{A}'}$$

Figure 3.3: Going to a location and performing an action results in two attack trees. The function *getAllPaths* returns all paths from the current locations of the actor to the goal location l, and the resulting attack trees are alternatives for reaching this location.

$$\frac{i \notin identities \implies \mathcal{T} = \mathcal{N}^{\text{obtain identity } i}}{\mathcal{I}, (identities, locations, assets), identity\ i \vdash_{credential} \mathcal{T}, (identities \cup \{i\}, locations, assets)}$$

$$\frac{\begin{array}{c}\mathcal{A} = (identities, locations, assets) \wedge a \notin assets \implies \\ goals = availableAt_{\mathcal{I}}(a) \qquad \mathcal{I}, \mathcal{A}, goals \vdash_{goal} trees, \mathcal{A}' \qquad \mathcal{T} := \oplus_{\vee}^{\text{“get a”}} trees\end{array}}{\mathcal{I}, \mathcal{A}, \textit{asset a} \vdash_{credential} \mathcal{T}, \mathcal{A}'}$$

$$\frac{\mathcal{I}, \mathcal{A}, \textit{predicate p(arguments)} \vdash_{predicate} trees, \mathcal{A}' \qquad \mathcal{T} := \oplus_{\vee}^{\text{“fullfil predicate } p\text{”}} trees}{\mathcal{I}, \mathcal{A}, \textit{predicate p(arguments)} \vdash_{credential} \mathcal{T}, \mathcal{A}'}$$

Figure 3.4: Depending on the missing credential, different attacks are generated. If the actor lacks an identity, an attack node representing an abstract social engineering attack is generated, for example, social engineering or impersonating. If the missing credential is an asset, the function *availableAt* returns a set of pairs of locations from which this asset is available, and the according $\mathbf{in}$ actions. If the missing credential is a predicate, a combination of credentials fulfilling the predicate must be obtained.

Attack Generation. The rules in Figure 3.2 connect the identified goals with the generation of attack trees. For each goal we generate two attack trees: moving to the location and performing the action. While moving to the location new credentials may be required; as a result, the actor acquires new knowledge, which is stored in the actor component $\mathcal{A}$. The rules in Figure 3.3 and Figure 3.4 generate attack trees for moving around, performing actions, and obtaining credentials, resulting in attack trees for every single action of the attacker. The resulting trees are combined in the overall attack tree. The function $missingCredentials$ uses the unification described above to match policies with the assets available in the model. This implies that all assets that can fullfil a policy are identified; the

attack generation then generates one attack for each of these assets, and combines them with a disjunctive node.

Post-Processing Attack Trees. The generated attack trees do not contain annotations or metrics about the success likelihood of actions such as social engineering, or the potential impact of actions. Also the likelihood of a given attacker to succeed or fail is not considered. Computing qualitative and quantitative measures on attack trees is beyond the scope of this work. The generated attack trees also often contain duplicated subtrees, due to similar scenarios being encountered in several locations, for example, the social engineering of the same actor, or the requirement for the same credentials. This is not an inherent limitation, but may clutter attack trees. Similar to (Vigo, Nielson, & Nielson, 2014), a post-processing of attack trees can simplify the result.

3.2 Generation of attacks through model-checking

Assuming that probability and cost of actions in a socio-technical security (STS) model are given, the attack generation approach described above can be made more precise as shown in (Lenzini et al., 2015). An overview of this approach is depicted in Figure 3.5. An STS model is mapped to a Probabilistic Symbolic Model Checker (PRISM) program. Alongside, security template expressions that define relevant security properties are instantiated as extended Probabilistic Computation Temporal Logic (PCTL) formulas. PRISM then takes as input the mapping of the STS model and the PCTL formulas, checks the satisfiability of security properties in the considered model, and produces the checking result in terms of probability and cost.

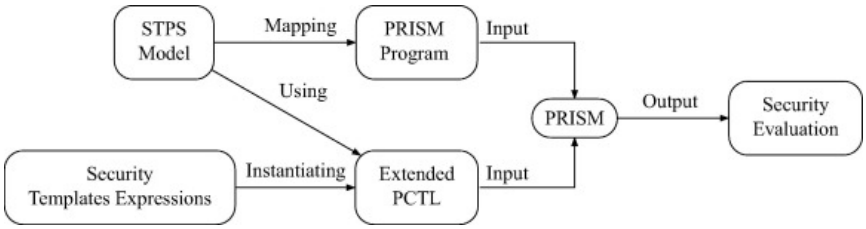


Figure 3.5: Scheme resuming the model-checking approach.

PRISM checks probabilistic specifications over probabilistic models. A specification can be expressed either in PCTL or in a continuous stochastic logic. A model can be described using the PRISM language as discrete-time Markov chains, continuous-time Markov chains, Markov decision processes (MDPs) or probabilistic (timed) automata. A PRISM program is a set of *modules*, each having a countable set of boolean or integer, local, variables. A module's local state is fully defined by the evaluation of its local variables, while the program's global state by the evaluation of all the variables.

Lenzini et al. encode an STS model into an equivalent PRISM program with Markov Decision Processes as the underlying formalism. The encoding is specified by five functions, namely Ψ_A, Ψ_O , Ψ_L, Ψ_I and Ψ_C. Given an STS model S, these functions return a PRISM

program P. Each function maps a fragment of the STS model into a PRISM module. Function Ψ_A encodes the honest actors and the transition rules that have an effect on them. Ψ_O encodes the objects and the transition rules that have an effect on objects. Ψ_L encodes the physical space, locations and doors, and the transition rules affecting them. These three functions together encode the structure which links actors, objects, locations and doors and the changes that operate on them by the STS transition rules. Besides, Ψ_I encodes the intruder and the transition rules of the intruder's actions, and Ψ_C encodes the transition costs. The final PRISM program is the composition, by synchronization, of the five modules.

Security properties are expressed in the extended Probabilistic Computation Tree Logic (PCTL), as it is able to express all the factors that the STS models describe, *i.e.* paths along locations, paths of actions, propositions on state variables, probabilities of occurrence of events and of sequences of events, and their costs. Formulas ϕ in this logic are generated by the following BNF grammar:

$$\begin{aligned} \phi &::= \top \mid ap \mid \phi \wedge \phi \mid \neg\phi \mid \mathrm{P}_{\bowtie p}[\psi] \mid \mathrm{R}_{\bowtie r}[F\phi] \\ \psi &::= \mathrm{X}\phi \mid \phi \mathrm{U} \phi \mid \phi \mathrm{U}^{\leq k} \phi \end{aligned}$$

Here, $k \in \mathbb{N}$, $r \in \mathbb{R}^+$, $p \in [0,1]$, and $\bowtie \in \{<, \leq, >, \geq\}$. A state formula can be "ap", an atomic proposition, or any propositional expressions built from formulas. $\mathrm{P}_{\bowtie p}[\psi]$, called *probabilistic path predicate*, returns true whether the probability to satisfy the *path formula* ψ, is $\bowtie\ p$ (*i.e.* less, no more, more, no less than p). $\mathrm{R}_{\bowtie r}[\phi]$, the *cost predicate*, returns true whether the cost to, eventually, reach a state satisfying ϕ is $\bowtie\ r$ (*i.e.* less, no more, more, no less than r). F is the modal operator *eventually*. A path formula is built from the temporal operators next (X), until (U), and bounded until ($\mathrm{U}^{\leq k}$). Other logic operators can be derived from the basic operators.

Besides, the *extended* PCTL has four more probabilistic and cost (quantitative) operators: Pmin, Pmax, Rmin, and Rmax. They can be used within path or a state formulas, and evaluate the minimum and the maximum probability and cost that the formula is true. More insight into PCTL can be found in (Forejt, Kwiatkowska, Norman, & Parker, 2011).

Three PCTL expression templates are used to specify STPS requirements:

1. $\phi := F\sigma$
2. $\phi' := \mathrm{Rmin}[\phi]$
3. $\phi'' := \mathrm{Pmax}[\phi]$

where σ is a predicate logic expression built from the three following atomic predicates: (a_o) (*i.e.* agents a holds object o), $(\mathsf{l}_a = i)$ (*i.e.* agent a is in location i), and $(\mathsf{l}_o = i)$ (*i.e.* object o is in location i).

The formula templates express reachability properties, in a socio-technical sense. They allow to check whether eventually an object ends to be in a location, or whether an actor or the intruder eventually reaches a location, or whether eventually he enters in possession of an object. They can be used to check for possibility of intrusions or of thefts. The

quantitative expressions ϕ' and ϕ'' are used to quantify, in terms of minimal cost and maximum probability, a security property.

3.3 Manual generation of attack scenarios

In case automated generation of attack scenarios is infeasible, e.g. if a reliable STS model is not available, the analyst still has the alternative to build up the security scenario in a systematic way. An approach of this type (Fraile et al., 2016) is what we introduce next. Our approach represents attack scenarios be means of ADTree models, and it is explained through the ATM case study (The TRESPASS Project, D7.4.2, 2016), whose goal is to capture the most dangerous multi-stage attack scenarios applicable to ATM structures.

Visualising the structure of the ADTree. ADtree is not only a formal mathematical language to capture potential attacks and defenses for a given system, but also a powerful communication means for security experts and stakeholders. It is thus convenient to structure the tree similarly to standard reports and documentation already familiar to stakeholders, lawyers, and analysts in general.

We observe that incident reports already present events and incidents within one or several categories. Typically, each event is of a particular type, and it is provided together with information about the attacker's goal and the attack steps. For this particular case study, we suggest the use of the EAST report 2015[1] and the ATMIA Global Fraud Survey 2015[2], which cover ATM fraud incidents in Europe. The implicit type relation between events in those reports allows us to create a taxonomy of attacks, which indeed resembles an attack tree (Figure 3.6).

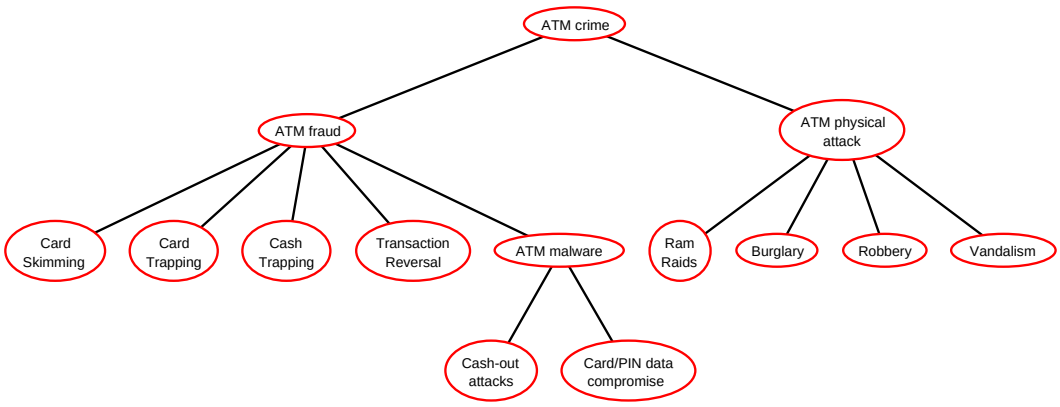


Figure 3.6: Taxonomy of ATM crime.

The taxonomy depicted in Figure 3.6 shows popular attacks against ATMs, which are first categorised into physical attacks and fraud-related attacks. Physical attacks require brute-force in order to tamper with the physical integrity of the ATM, while ATM fraud attacks

[1] https://www.european-atm-security.eu/tag/european-atm-crime-report/
[2] https://www.atmia.com/whitepapers/global-fraud-survey-2015/1104/

are more sophisticated and involve compromising credentials of legitimate accounts via a variety of techniques, such as card skimming or malware (malicious software) infection.

Overcoming the lack of attack intelligence. The task of mapping a security scenario into an attack tree greatly depends on the security expertise of the team developing the attack tree. However, security expertise needs to be complemented with data about previous attacks. Such data can come in the form of an attack pattern library[3], i.e. a structure containing precondition and postcondition of attacks, attack profiles, and a glossary of defined terms and phrases. Yet, businesses and governments are usually reluctant to disclose attack data, as it may harm their reputation and could help attackers to exploit similar vulnerabilities.

To overcome this problem we propose the use of different sources of information. In the case of ATM security, we consider *PCI-DSS ATM Security Guidelines*[4] to understand how secure channels for payment systems based on smartcards are implemented, *ATM Industry Association*[5] to collect best practices in ATM security, *EU law enforcement agency*[6] to depict recent trends in cybercrime, *National Crime Agency*[7] and *ATM Marketplace*[8] to gather recent reports on financial fraud and potential countermeasures. This study results in a better understanding of the above taxonomy, therefore leading to a more fine-grained tree with countermeasures, i.e. to an actual ADTree as depicted in Figure 3.7. We remark that, due to space constraints, Figure 3.7 only shows extracts of the full ADtree.

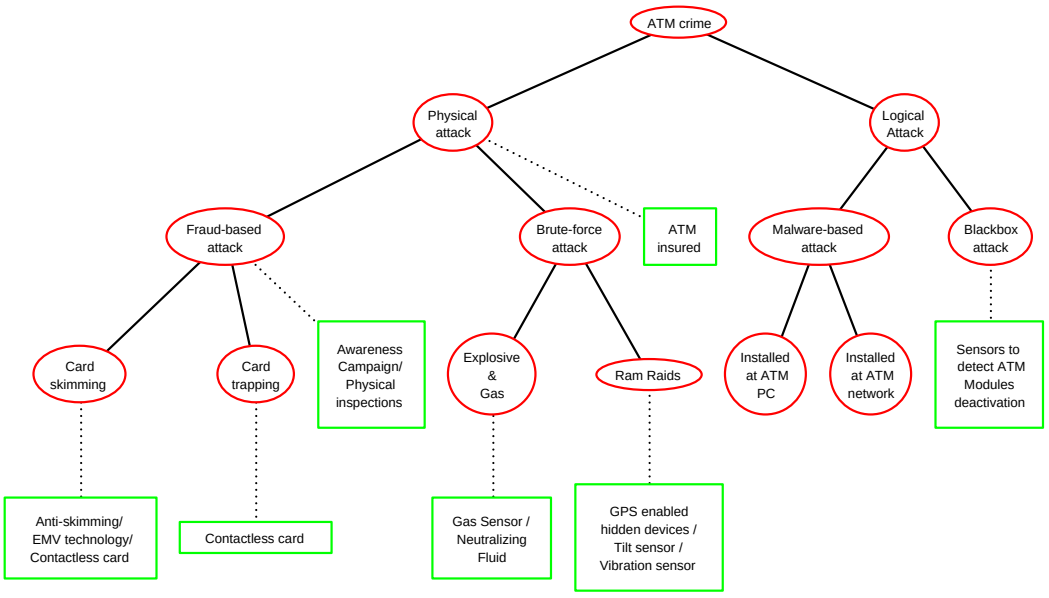


Figure 3.7: An excerpt of an ADTree on ATM crime.

[3] www.sei.cmu.edu/reports/01tn001.pdf
[4] https://www.pcisecuritystandards.org/pci_security/
[5] https://www.atmia.com/
[6] https://www.europol.europa.eu/
[7] http://www.nationalcrimeagency.gov.uk/
[8] http://www.atmmarketplace.com/

A visible improvement of the model in Figure 3.7 is the presence of countermeasures. For example, card skimming and cash trapping can be prevented by anti-skimming solutions such as stress sensors, or by making compulsory the use of EMV technology (Chip&PIN) and contactless cards. A general countermeasure against this type of fraud is to make customers and employees aware of the fraud in order to perform quick physical inspections themselves. Brute force attacks in contrast, cannot be actually prevented, but detected. Detection mechanisms are GPS-enabled devices to localize the ATM, tilt, vibration, and gas sensors, etc. A fairly recent commercially available prevention technique against gas attacks is *neutralizing fluid*, which is expected to delay the gas explosion.

Capturing attack vectors in a semantically meaningful way. The ADTree we produced, an excerpt of which is shown in Figure 3.7, is a useful classification of attacks to ATMs, and applicable countermeasures and mitigation strategies. However, it does not benefit from the main feature of ADTrees as a mathematical language, that is, the ability to encode several attack vectors in a compact tree structure. An *attack vector* is a path or a set of attack steps an adversary can follow in order to successfully attack a system. In ADTrees, attack vectors are expressed by using the conjunctive operator AND, which expresses that all sub-goals of a given goal ought to be achieved.

The main challenge when creating a large ADTree is to guarantee that it is semantically meaningful, while keeping its communication potential. We propose to address this challenge by frequently executing two different verification processes. The first one consists in checking that the taxonomies depicted in Figures 3.6 and 3.7 are preserved as much as possible. The second one consists in keeping track of those attack vectors we expect to model, and verifying that the multiset semantics of ADTrees (B. Kordy, Mauw, Radomirović, & Schweitzer, 2010) matches this set of attack vectors. The result of this process is a comprehensive attack model for the ATM case study, which can be downloaded from `http://satoss.uni.lu/members/piotr/adtool/`.

Summary of lessons learned. We have found it very helpful starting to create an ADTree from a taxonomy of attacks. This established attack taxonomy allows us to structure the reasoning and compare the identified attacks with globally known attacks, thus serving as a reference to check the tree for completeness. In general, we also have found that several countermeasures did not really prevent the attack, but triggered actions that could mitigate the impact of the attack. Handling these countermeasures required lengthy team discussions. We suspect that these challenges in handling treatment options arise from the fact that they are not clearly addressed in the ADTree methodology itself (e.g. any of the established semantics). One possibility is to extend the ADTree methodology by explicitly typing defense nodes, following the example of attack-countermeasure trees that support detective and reactive countermeasures (Roy et al., 2012b). However, this change will also increase the cognitive load on the analysts.

4 Ranking

Human ability to visualise and understand attack models quickly decreases with the increase in size and complexity of the model. Identifying important portions of a model is therefore of paramount importance for security analysts; it allows us to prioritize and focus on parts that contribute most to the attacker goal. A systematic approach to prioritization is *ranking*, whereby a set of elements is sorted with respect to a total or partial order.

The tools developed in TREsPASS enable not only the prediction of attack values and attack paths, thereby identifying serious vulnerabilities in the system, but also *prioritisation and ranking of attacks*. This is relevant in the security landscape, where enterprises have limited budgets and want to select countermeasures that provide the best trade-offs between the impact of perceived risk and the costs for implementing it.

In this section we start by briefly introducing Pareto frontier, a technique that allows ranking to be performed when a total order is not available. We also explain approaches based on a total order, such as ranking on priced automata and attack-defence trees.

4.1 Pareto frontier: ranking in attack trees

In many real-life scenarios multiple objectives might be required for analysis of complex attack scenarios. In queries with multiple objectives, in particular with conflicting objectives, there is no best solution but rather a set of efficient solutions, known as Pareto frontier. In order to analyse complex scenarios with multiple objectives, we devise automated techniques, described in (The TRESPASS Project, D3.3.2, 2015), that optimise all objectives at once and compute the set of all optimal solutions, defined in terms of Pareto efficiency. The resulting Pareto frontier can be further analysed and the associated attacks can be ranked based on a particular objective. For example, given a cost threshold for an attacker we can eliminate the attacks that are outside of the cost threshold and rank the most successful attacks based on the threshold.

4.2 Model checking priced automata

Ranking can be also performed by model checking timed automata using UPPAAL CORA. This model checker has the inbuilt *best trace option*, which can be used to find an optimal trace (Behrmann, Larsen, & Rasmussen, 2005). 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. To find different attacks ranked according to their cost, we repeat the procedure above, each time excluding the attack paths we have already found. Thus, we can determine the top-10 worst attacks with respect to different metrics (minimum cost, minimum time, etc.)

The ranking process described above relies on the standard decoration process for attack trees. That is, a process where attack tree leaves are enriched with attribute values, such as time, skill, and resources. These components can be dependent, e.g. time can depend on the skill level. Then our analysis techniques map these cost structures semantically on priced timed automata as an underlying mathematical model in a compositional manner.

4.3 Efficient bottom-up ranking in attack-defence trees

In attack graphs, a modelling language similar to attack trees, several ranking approaches have been defined (Mehta, Bartzis, Zhu, Clarke, & Wing, 2006). In attack-defence trees, however, ranking has been mostly neglected by both quantification methods and tools. The `ADTool2.0` implements an efficient and formal approach to rank attack scenarios. In particular, we have extended the bottom-up computation approaches proposed for attack trees (Mauw & Oostdijk, 2005), attack-defense trees (B. Kordy, Mauw, Radomirović, & Schweitzer, 2010), and `SAND` attack trees (Jhawar et al., 2015), in order to efficiently rank attack scenarios, where an attack scenario is either a *bundle* as in the formalisms in (Mauw & Oostdijk, 2005; B. Kordy, Mauw, Radomirović, & Schweitzer, 2010) or a *Series-Parallel (SP) graph* as in (Jhawar et al., 2015). Our approach works intuitively as follows. Given a set of quantitative values V for attack scenarios and a total order $\leq$ on V, we store at every node of the tree n least attacks with respect to the total order $\leq$, where n is a natural number representing a bound on the number of attack scenarios to be ranked. We illustrate this approach formally in `SAND` trees.

We write $\xrightarrow{b}$ for the SP graph with a single edge labeled with b and define SP graphs as follows.

Definition 1. *The set $\mathbb{G}_{SP}$ of* series-parallel graphs *(SP graphs) over a set of basic actions $\mathbb{B}$ is defined inductively by the following two rules*

- *For $b \in \mathbb{B}$, $\xrightarrow{b}$ is an SP graph.*
- *If G and G' are SP graphs, then so are $G \cdot G'$ and $G \parallel G'$.*

The SP semantics for `SAND` attack trees is defined as follows.

Definition 2. *The* SP semantics *for* SAND *attack trees is given by the function* $[\![\cdot]\!]_{\mathcal{SP}} : \mathbb{T}_{\texttt{SAND}} \to \mathcal{P}(\mathbb{G}_{\mathcal{SP}})$*, which is defined recursively as follows: for* $b \in \mathbb{B}$*,* $t_i \in \mathbb{T}_{\texttt{SAND}}$*,* $1 \leq i \leq k$*,*

$$
\begin{aligned}
&[\![b]\!]_{\mathcal{SP}} = \{\overset{b}{\to}\} \\
&[\![\mathit{OR}(t_1, \ldots, t_k)]\!]_{\mathcal{SP}} = \textstyle\bigcup_{i=1}^{k} [\![t_i]\!]_{\mathcal{SP}} \\
&[\![\mathit{AND}(t_1, \ldots, t_k)]\!]_{\mathcal{SP}} = \{G_1 \parallel \ldots \parallel G_k \mid (G_1, ..., G_k) \in [\![t_1]\!]_{\mathcal{SP}} \times ... \times [\![t_k]\!]_{\mathcal{SP}}\} \\
&[\![\texttt{SAND}(t_1, \ldots, t_k)]\!]_{\mathcal{SP}} = \{G_1 \cdot \ldots \cdot G_k \mid (G_1, ..., G_k) \in [\![t_1]\!]_{\mathcal{SP}} \times ... \times [\![t_k]\!]_{\mathcal{SP}}\}.
\end{aligned}
$$

An *attribute domain for an attribute* A_α *on* SAND *attack trees* is a tuple $D_\alpha = (V_\alpha, \triangledown_\alpha, \triangle_\alpha, \Diamond_\alpha)$, where V_α is a set of values and $\triangledown_\alpha, \triangle_\alpha, \Diamond_\alpha$ are families of k-ary functions of the form $V_\alpha \times \cdots \times V_\alpha \to V_\alpha$, associated to OR, AND, and SAND refinements, respectively. An *attribute for* SAND *attack trees* is a pair $A_\alpha = (D_\alpha, \beta_\alpha)$ formed by an attribute domain D_α and a function $\beta_\alpha : \mathbb{B} \to V_\alpha$, called *basic assignment* for A_α, which associates a value from V_α with each $b \in \mathbb{B}$.

Definition 3. *Let* $A_\alpha = \big((V_\alpha, \triangledown_\alpha, \triangle_\alpha, \Diamond_\alpha), \beta_\alpha\big)$ *be an attribute. The attribute evaluation function* $\alpha : \mathbb{T}_{\texttt{SAND}} \to V_\alpha$*, which calculates the value of attribute* A_α *for every* SAND *attack tree* $t \in \mathbb{T}_{\texttt{SAND}}$*, is defined recursively as follows.*

$$
\alpha(t) = \begin{cases}
\beta_\alpha(t) & \text{if } t = b,\ b \in \mathbb{B} \\
\triangledown_\alpha\big(\alpha(t_1), \ldots, \alpha(t_k)\big) & \text{if } t = \mathit{OR}(t_1, \ldots, t_k) \\
\triangle_\alpha\big(\alpha(t_1), \ldots, \alpha(t_k)\big) & \text{if } t = \mathit{AND}(t_1, \ldots, t_k) \\
\Diamond_\alpha\big(\alpha(t_1), \ldots, \alpha(t_k)\big) & \text{if } t = \texttt{SAND}(t_1, \ldots, t_k)
\end{cases}
$$

An important property of an attribute is *compatibility*, whereby every pair of semantically equivalent SAND attack trees have equal attribute value.

Definition 4. *An attribute* A_α *is said to be compatible with the SP semantics if and only if for all* SAND *attack trees* t *and* t'*, we have*

$$
[\![t]\!]_{\mathcal{SP}} = [\![t']\!]_{\mathcal{SP}} \implies \alpha(t) = \alpha(t').
$$

The attribute valuation function can also be defined over the universe of SP graphs by $\alpha(G) = \alpha(t)$ where t is a SAND attack tree whose semantics is $\{G\}$. The attribute valuation function can also be defined over the universe of SP graphs by

$$
\alpha(G) = \begin{cases}
\alpha(b) = \beta_\alpha(b) & \text{if } G = \overset{b}{\to} \\
\triangle_\alpha\big(\alpha(G_1), \alpha(G_2)\big) & \text{if } G = G_1 \parallel G_2 \\
\Diamond_\alpha\big(\alpha(G_1), \alpha(G_2)\big) & \text{if } G = G_1 \cdot G_2
\end{cases}
$$

For a set of SP graphs $X = \{G_1, \ldots, G_k\}$, we set

$$
\alpha(X) = \triangledown(\alpha(G_1), \ldots, \alpha(G_k)).
$$

For $t \in \mathbb{T}_{\texttt{SAND}}$, we thus obtain

$$
\alpha(t) = \alpha([\![t]\!]_{\mathcal{SP}}).
$$

Definition 5 (Ranking function)**.** *Let* $A_\alpha = \big((V_\alpha, \triangledown_\alpha, \triangle_\alpha, \Diamond_\alpha), \beta_\alpha\big)$ *be an attribute, and* $\leq$ *a total order over* V_α*. We say that a function* $r_n : \mathbb{T}_{\texttt{SAND}} \to \mathbb{G}^*_{\mathcal{SP}}$ *with* $n > 1$ *is a* ranking function *with respect to* $(A_\alpha, \leq)$ *if for every* SAND *attack tree* $t \in \mathbb{T}_{\texttt{SAND}}$*,* $r_n(t) = (G_1, \ldots, G_m)$ *implies that:*

- $m = \min\{n, |[\![t]\!]_{\mathcal{SP}}|\}$
- $G_i \in [\![t]\!]_{\mathcal{SP}}\ \forall i \in \{1, \ldots, m\}$
- $\alpha(G_1) \leq \alpha(G_2) \cdots \leq \alpha(G_m)$
- *For every* $G \in [\![t]\!]_{\mathcal{SP}}$ *either* $G \in r_n(t)$ *or* $\alpha(G_m) \leq \alpha(G)$

A ranking function can be computed straightforwardly as follows. Given a SAND attack tree t, consider all SP graphs $\{G_1, \ldots, G_m\}$ in the semantics of t. The set of SP graphs is re-ordered as $\{G_{i_1}, \ldots, G_{i_m}\}$ in such a way that $G_{i_j} \leq G_{i_{j+1}}\ \forall j \in \{1, \ldots, m-1\}$. The output of the ranking function r_n is $(G_{i_1}, \ldots, G_{i_m})$ if $m \leq n$, $(G_{i_1}, \ldots, G_{i_n})$ otherwise.

The problem with the above computation is that it requires the computation of the set of SP graphs in the semantics of a SAND attack tree, while such a set can be of exponential size in terms of the number of leaf nodes in the tree. Therefore, we propose next a more efficient approach that draws upon the recursive definition of the valuation function α. We first define, given an attribute A_α, a *ranked-union* operator $\cup_n$ between sequences of SP graphs as follows: $(G'_1, \ldots, G'_{m_1}) \cup_n (G''_1, \ldots, G''_{m_2}) = (G_1, \ldots, G_m)$ where:

- $m = \min\{n, \max\{m_1, m_2\}\}$
- For every $i \in \{1, \ldots, m\}$ there exists $j \in \{1, \ldots, \max\{m_1, m_2\}\}$ such that either $G_i = G'_j$ or $G_i = G''_j$
- $\alpha(G_1) \leq \alpha(G_2) \cdots \leq \alpha(G_m)$
- For every $i \in \{1, \ldots, m_1\}$ (resp. $i \in \{1, \ldots, m_2\}$) either $G'_i = G_j$ (resp. $G''_i = G_j$) for some $j \in \{1, \ldots, m\}$ or $G_m \leq G'_i$ (resp. $G_m \leq G''_i$)

Intuitively, a ranked-union operator $\cup_n$ simply join two sequences of SP graphs while respecting the order imposed by the total order $\leq$ and keeping at most n SP graphs. We observe that, given a SAND attack tree and its SP semantics $\{G_1, \ldots, G_m\}$, the function $r_n = (G_1, \ldots, G_m) \cup_n \emptyset$ is indeed a ranking function.

We use the ranked-union to define two other operators: the *ranked-parallel-product* $\times^{\|}_n$ and *ranked-sequential-product* $\times^{\cdot}_n$ operator.

- $(G'_1, \ldots, G'_{m_1}) \times^{\|}_n (G''_1, \ldots, G''_{m_2}) = (G_1, \ldots, G_{m_1 \times m_2}) \cup_n \emptyset$ where $G_i = G_{i \mod m_1} \| G_{i \% m_2}$ for every $i \in \{1, \ldots, m_1 \times m_2\}$.
- $(G'_1, \ldots, G'_{m_1}) \times^{\cdot}_n (G''_1, \ldots, G''_{m_2}) = (G_1, \ldots, G_{m_1 \times m_2}) \cup_n \emptyset$ where $G_i = G_{i \mod m_1} \cdot G_{i \% m_2}$ for every $i \in \{1, \ldots, m_1 \times m_2\}$.

Both operators $\times^{\|}_n$ and $\times^{\cdot}_n$ are extended from binary to k-ary by considering the left-associativity property of both $\|$ and $\cdot$.

Definition 6 (Bottom-up ranking function)**.** *Let* $A_\alpha = \big((V_\alpha, \triangledown_\alpha, \vartriangle_\alpha, \lozenge_\alpha), \beta_\alpha\big)$ *be an attribute. The ranking function* $f_n : \mathbb{T}_{\texttt{SAND}} \to \mathcal{P}(\mathbb{G}_{\mathcal{SP}})$ *is defined recursively as follows.*

$$f_n(t) = \begin{cases} (\{\overset{b}{\rightarrow}\}) & \text{if } t = b,\ b \in \mathbb{B} \\ f_n(t_1) \cup_n \cdots \cup_n f_n(t_k) & \text{if } t = \textit{OR}(t_1, \ldots, t_k) \\ f_n(t_1) \times_n^{\|} \ldots \times_n^{\|} f_n(t_k) & \text{if } t = \textit{AND}(t_1, \ldots, t_k) \\ f_n(t_1) \times_n^{\cdot} \ldots \times_n^{\cdot} f_n(t_k) & \text{if } t = \texttt{SAND}(t_1, \ldots, t_k) \end{cases}$$

Proposition 1. *The bottom-up ranking function is a ranking function as defined in Definition 6.*

We expect the attribute value of the top ranked SP graph to be equal to the attribute value of the SAND attack tree. We formalize this definition as follows.

Definition 7. *Let* $A_\alpha = \big((V_\alpha, \triangledown_\alpha, \vartriangle_\alpha, \lozenge_\alpha), \beta_\alpha\big)$ *be an attribute and* $\leq$ *a total order on* V_α*. The ranking function* $r_n : \mathbb{T}_{\texttt{SAND}} \to \mathcal{P}(\mathbb{G}_{\mathcal{SP}})$ *with respect to* (A_α, V_α) *is* sound *if for every* SAND *attack tree* t*, we have*

$$f_n(t) = (G_1, \ldots, G_m) \implies \alpha(t) = \alpha(G_1).$$

Proposition 2. *Let* $A_\alpha = \big((V_\alpha, \triangledown_\alpha, \vartriangle_\alpha, \lozenge_\alpha), \beta_\alpha\big)$ *be an attribute and* $\leq$ *a total order on* V_α*. The bottom-up ranking function* $f_n : \mathbb{T}_{\texttt{SAND}} \to \mathcal{P}(\mathbb{G}_{\mathcal{SP}})$ *with* (A_α, V_α) *is sound if the following two conditions are satisfied:*

- $\triangledown_\alpha$ *is an extrema function providing the least element within the input set, i.e.,* $\triangledown_\alpha(x_1, \ldots, x_k) = y \implies y \in \{x_1, \ldots, x_k\} \wedge y \leq x_i\ \forall i \in \{1, \ldots, k\}$
- $\vartriangle_\alpha$ *and* $\lozenge_\alpha$ *is monotonic, i.e., for every* $(x_1, \ldots, x_k)$ *and* $(y_1, \ldots, y_k)$*,* $x_i \leq y_i \forall i \in \{1, \ldots, k\} \implies \vartriangle_\alpha (x_1, \ldots, x_k) \leq \vartriangle_\alpha (y_1, \ldots, y_k) \wedge \lozenge_\alpha(x_1, \ldots, x_k) \leq \lozenge_\alpha(y_1, \ldots, y_k)$

We observe that most attribute domains supported by the ADTool2.0, indeed, satisfy these two properties.

4.3.1 Ranking in the ADTool2.0.

This ranking approach has been implemented in the ADTool2.0. It can be found in the *Ranking View* window, which can be opened from the menu Windows → Ranking View. As in the Attribute window, the Ranking window gives the option to open or create an attribute domain. By default, the ADTool2.0 uses as a total order the operator assigned to the OR gate in the attribute domain. A screenshot of the ADTool2.0 provided in Fig. 4.1 shows an example of the ranking feature applied to a SAND attack tree modelling the Stuxnet attack (inspired by (Kriaa, Bouissou, & Pièvre-Cambacédès, 2012)).

In order to rank attack scenarios up to a given node in the tree, we ought to click that node in the domain for which we want to see the ranking. Doing so, the Ranking view window will automatically update with a table containing optimal attacks with respect to the

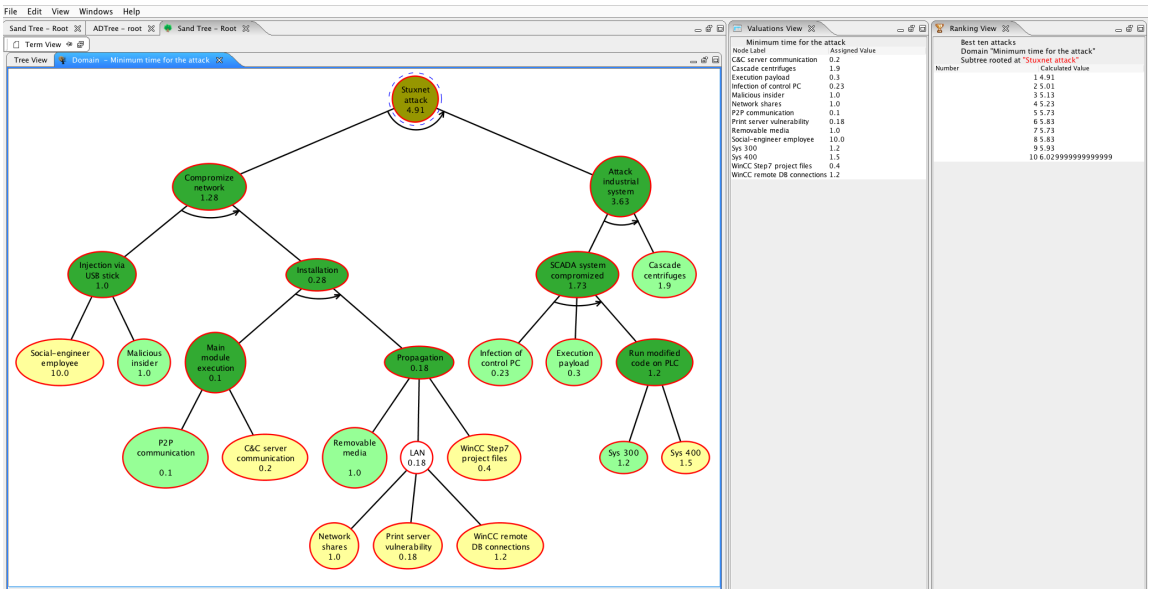


Figure 4.1: Screenshot of the `ADTool2.0` with the ranking feature. The `SAND` attack tree used represents the Stuxnet attack, and the ranking is based on the minimal time of attack parameter. The attack scenario (all its attack nodes) with the minimal time of execution is highlighted in green by the tool.

chosen attribute domain. The `ADTool2.0` also offers the option to highlight those nodes that contribute most to the attack, which can be done by clicking on attack scenarios in the ranking table.

5 Preventive measures

After possible attacks have been generated (Chapter 3) and the most critical attacks have been identified (Chapter 4), the TRESPASS analysis process includes a step to recommend countermeasures that can be introduced in the system in order to prevent critical attacks and limit their impact. While it is not yet clear if the countermeasure selection process can be fully automated, it should be at least tool-assisted. In this section we provide two approaches to assist human experts in selecting the right security controls.

The first approach automatically generates attack-defence trees from the TRESPASS socio-technical model. These generated attack-defence trees include security mechanisms available in the socio-technical model (the access control mechanisms) explicitly as defence nodes (The TRESPASS Project, D3.4.1, 2014; Gadyatskaya, 2015). The second approach, instead, is a comprehensive methodology to enrich an attack tree with countermeasures (Gadyatskaya, Harpes, Mauw, Muller, & Muller, 2016). This approach is supported by a tool called *ADTop*, which can help organisations to fine-tune their risk assessments and build their own libraries of attacks and countermeasures in the real-world situation. ADTop actually extends the TRICK Service, and has been validated thanks to Epstan, a realistic case study documented in (The TRESPASS Project, D7.4.2, 2016). Finally, we introduce a novel approach to generate *Attack-Defence Diagrams* (Hermanns et al., 2016).

5.1 Automated generation of attack-defence trees

In Section 3.1 we have presented our approach to generate attack trees from the TRESPASS socio-technical model. This approach has been also extended in the project to generate attack-defence trees. The attack-defence tree formalism includes not only attack nodes, but also countermeasure nodes (see Section 2.3 for more details). Therefore, it is practical to generate attack-defence trees from the socio-technical model as this formalism allows us to automatically capture countermeasures already present in the (modelled part of) infrastructure, and to assist the analyst in subsequent reasoning for new countermeasures based on quantitative analysis on attack-defence trees. The approach presented in this section is published in (Gadyatskaya, 2015), and the present section is a summary of this paper.

5.1.1 Extraction of defences from the model

The only security controls captured in the TRE$_S$PASS socio-technical model (see (The TRE$_S$PASS Project, D1.3.4, 2016) for more details of the model) captures are access control policies that restrict access to certain locations. These policies roughly correspond to physical (locks) or digital (password check) means (policy enforcement mechanisms) implemented in the system to restrict access to assets. Therefore, we make explicit in the attack-defence tree generation process the fact that the attacker needs to overcome the restrictions imposed by security policies. To achieve that we will use attack-defence bundles that are based on the attack-defence tree formalism (B. Kordy, Mauw, Radomirović, & Schweitzer, 2012b).

Intuitively, the attacker has two options for dealing with security policies in the system. He can attempt to satisfy the access control policy (for example, by collecting the necessary credentials or coercing someone with the right credentials) or he can try to circumvent the policy (e.g., by forcing the lock). The first approach is in line with the attack tree generation by policy invalidation process (see Section 3.1 for more details of this core TRE$_S$PASS technique), because it can be automatically designed based on reachability. If we want to refine the second approach, we need to understand how exactly different policies (more precisely – enforcement mechanisms for these policies) can be circumvented. There is a need to represent the human expert knowledge in circumventing different security mechanisms in such a way that it is useful for automated generation process. This can be achieved, for example, by applying the attack patter library mechanism (see (The TRE$_S$PASS Project, D5.4.2, 2016) for more details).

Indeed, the enforcement mechanisms for access control policies defined in the socio-technical model can be automatically captured into attack-defence trees. If the knowledge about breaking certain kinds of enforcement mechanisms is available in a suitable format (e.g. as an attack subtree), then the attack-defence trees can be further refined based on that information. Further analysis based on the attack-defence trees produced at this stage (e.g., computation of the most probable or the most cheap attack for the attacker) can identify the missing enforcement mechanisms. For example, if it turns out that the attacker can directly access an asset because no access control policy is enforced for this asset, it might be the first recommendation for improving security of the organisation: to introduce an appropriate access control mechanism to protect access to the asset.

5.1.2 Generation of attack-defence bundles

We introduce a simplified TRE$_S$PASS socio-technical model to exemplify the attack-defence model creation. The simplified model allows to reason only about potential reachability. However, this is already very useful for risk analysis, as quantitative evaluation of the possibility that an attacker accesses some system elements can simplify risk analysis for human analysts (Othmane, Ranchal, Fernando, Bhargava, & Bodden, 2015).

The simplified model captures simultaneously organisation's infrastructure topology for both physical and digital locations, as well as actors moving around this infrastructure

(these can be persons or processes). In the model these entities are represented as a set of model elements N that is a union of a set of infrastructure locations N_i, actors N_a, and objects N_o. We consider two domains: Ph is the physical space (model elements in this domain are physical entities, including, e.g. rooms, persons, and items), while Dg is the digital space (network locations and processes are in this domain), such that $N = Ph \cup Dg$, and $Ph \cap Dg = \emptyset$.

Some model elements are connected. We denote as $E \subseteq N \times N$ the set of directed connections. All edges e in E are of the following types:

- $e \in E_{ii} \subseteq N_i \times N_i$: connections between infrastructure locations (rooms, corridors, etc.). These connections are assumed bi-directional. More precisely, if $(i_1, i_2) \in E_{ii}$ then $(i_2, i_1) \in E_{ii}$.
- $e \in E_{ai} \subseteq N_a \times N_i$: placement of actors in the infrastructure;
- $e \in E_{oi} \subseteq N_o \times N_i$: placement of objects in the infrastructure;
- $e \in E_{oa} \subseteq N_o \times N_a$: placement of objects that are carried around by actors;
- $e \in E_{oo} \subseteq N_o \times N_o$: placement of objects that are inside other objects; here $e = (o_1, o_2)$ denotes an object o_1 located within an object o_2.

Mutual intersections of $E_{ii}, E_{ai}, E_{oi}, E_{oa}, E_{oo}$ are empty sets. Elements of the same domain can be connected liberally. However, some self-evident restrictions apply when connections between elements of the physical and digital domains are considered. For example, a data file cannot be located in an office or inside a cupboard. We allow multiple locations for the same actor and object. This corresponds to the possibility of actors to move in the model, and represents that some items can appear in several locations.

We define a location function $\texttt{loc}()$: $N \times N$ as follows: $\forall n \in N\ \texttt{loc}(n)$: = $\{l \in N | (n, l) \in E\}$.

Notice that for infrastructure locations or actors the function $\texttt{loc}()$ returns infrastructure locations where these model items are accessible from. However, as objects can be accessible from actors or other objects, $\texttt{loc}()$ may return any type of items in the model.

Policies. Let P be a set of policies defined in the model. We consider access control policies represented as tuples restricting access to element n. The *local policy* δ_n is a set of individual access control configurations. Each access control configuration $p \in \delta_n$ is a tuple $\langle Cred, atLocation, EM \rangle$, where $Cred \subseteq N_o$ is a set of credentials required to get access, $atLocation \in N$ s.t. $(n, atLocation) \in E$ is a model element from which access to n is granted, and $EM \in N$ is a reference to the mechanism enabled in the model to enforce the policy. EM can be the same as $atLocation$, meaning that the enforcement mechanism is implemented right at the spot (e.g., a lock), it can be an actor (e.g., a security guard checking identity documents or a process implementing access control), or an object. Notice that we assume that $c \in Cred \subseteq N_o$ is an asset present in the model, which can be either an item or data.

In theory, different access control configurations of the same local policy δ_n can be enforced by different enforcement mechanisms. For example, to access a building employees might use a badge applying it to an RFID-reader, or they might show their IDs to a security guard.

We will now show how to generate *attack-defence bundles* (AD-bundles) that can be used to capture the attack-defence state of the system. AD-bundles are generated for individual assets. They consist of attack nodes that correspond to gaining access to items in the model and attacking these items, and defence nodes that represent protections offered by the local policies in place. Notice that the bundles are attacker-agnostic, and they refer only to the system configuration regarding some particular item. Our notation abuses the standard notation for attack-defence trees, as we use AD-terms to represent both the tree structure and to refer to concrete attacker goals. We also define different types of AD-terms. This is syntactic sugar to ease the type representation, as types are used to put bundles together and synthesise AD-trees.

Attack node types. We consider attack nodes can be of the following types.

- $access_n$ is an attack node that represents that the attacker gains access to item n.
- $access\_from_{n,l}$ represents the goal of the attacker to access item n from specific model element l. This node type explicitly states the way n is accessed in the model, thus allowing us to understand immediately what access control policy is applicable (by looking at the $atLocation$ attribute).
- $break_n$ represents the goal of the attacker to somehow disable an access control mechanism implemented in n (this enforcement mechanism can protect assets not located in n).
- $attack\_pol_p$ represents the goal of the attacker to overcome protection of an individual access control configuration p.
- sat_pol_p represents attacker's goal to satisfy access control configuration p (by collecting all necessary credentials).

Defence node types. The defence nodes can be of the following types:

- $EM_{n,l}$ represents the defence of enforcement mechanisms enforcing policies at l to control access to n (notice that the enforcement mechanism itself can be located elsewhere).
- pol_config_p represents protection offered by an individual access control configuration for some $p \in \delta_n$.

Notice that term types $attack\_pol$ and pol_config are required to satisfy the requirement of AD trees for the unique child of the opposite type (B. Kordy et al., 2012b).

Bundle construction. Let $n \in N$ be an item in the model. An AD-bundle $\mathcal{B}_n$ that characterises accessing n is constructed as follows.

We start by setting the root of the bundle to $access_n$, as this is the desired attacker's goal.

Next, $access_n$ is refined:
$access_n$:= $\vee^p \Big(access\_from_{n,l} | l \in \texttt{loc}(n)\Big)$ // *n can be accessed only from an adjacent element in the model. Any of these elements is suitable for the attacker*

If $\nexists p = \langle Cred, l, EM\rangle \in \delta_n$ then $access\_from_{n,l}$:= $access_l$ // *Access to n from l can be implemented by simply accessing l. No access control policy is set up to guard this connection.*

If $\exists p = \langle Cred, l, EM\rangle \in \delta_n$ then $access\_from_{n,l}$:= $c^p \Big(access_l, EM_{n,l}\Big)$ // *Access to n from l can be implemented by accessing l. However, as there is an enforcement mechanism that controls access, the defence node is also added.*

$EM_{n,l}$:= $\wedge^o \Big(pol\_config_p | \forall p \in \delta_n$ s.t. $p = \langle Cred, l, s\rangle\Big)$ // *Protection of access from l to n is implemented via individual policy configurations.*

pol_config_p := $c^p \Big(attack\_pol_p\Big)$ // *syntactic sugar to switch back to attacker's view*

$attack\_pol_p$:= $\vee^p \Big(sat\_pol_p, break_s\Big)$, where $p = \langle Cred, l, s\rangle$ // *Attacker can either satisfy the individual policy configuration p, or he can break the enforcement mechanism s that enforces this configuration p.*

sat_pol_p := $\wedge^p \Big(access_{cred} | \forall cred \in Cred\Big)$, where $p = \langle Cred, l, s\rangle$ // *To satisfy the configuration the attacker needs to access all credentials in the set $Cred$ identified in this configuration.*

5.1.3 Approach to synthesise attack-defence trees

AD-bundles represent attacks on individual assets in the model. They can be "glued" together to form AD-trees, in the spirit of attack generation by policy invalidation. In this subsection we outline an approach to synthesis of attack-defence trees.

The main requirement for AD-trees synthesis is that it should terminate. Indeed, it is easy to see that any simple loop in the infrastructure will create infinite trees if bundles are composed naively. Moreover, some bundles may appear more than once in the generated tree, creating duplicate subtrees. To avoid this, we introduce a system state that will keep track of already achieved progress and will allow to terminate the synthesis process when the attacker has achieved the goal.

We define now two functions that identify the state of the system. These functions will be updated as the attack tree is generated in order to keep track with the attack development.

Definition 8 (`Reachable(,)`)**.** *Let $\mathcal{M} = (N, E)$ be a model. We define a boolean function* `Reachable`$(,) \subseteq N_a \times N$*:*

- *If $(a, n) \in E$,* `Reachable`(a, n) *:=* `True`*.*

- *If for some* $l \in N_i$ $(a, l) \in E_{ai}$ *and* $(o, l) \in E_{oi}$*, then* $\texttt{Reachable}(a, o)$ *:=* $\texttt{True}$*.*
- *If for some* $l \in N_i$ $(a, l) \in E_{ai}$ *and* $(a_1, l) \in E_{ai}$*, then* $\texttt{Reachable}(a, a_1)$ *:=* $\texttt{True}$ *and* $\texttt{Reachable}(a_1, a)$ *:=* $\texttt{True}$*.*
- *Else* $\texttt{Reachable}(a, n)$ *:=* $\texttt{False}$*.*

This function initially captures for a given actor all items immediately reachable in the model. These items can be objects or actors located in the same location as the actor. Let $\texttt{Reach}(a)$:= {$\forall n \in N$ s.t. $\texttt{Reachable}(a, n)$ = $\texttt{True}$}.

Definition 9 ($\texttt{Granted}(,)$)**.** *We define a boolean function* $\texttt{Granted}(,) \subseteq N_a \times N$*:*

- *If for an item* n $\delta_n = \emptyset$ *then* $\texttt{Granted}(a, n)$ *:=* $\texttt{True}$*.*
- *If for an item* n *there is a tuple* $p = \langle Cred, atLocation \rangle \in \delta_n =$ *s.t.* $Cred \subseteq \texttt{Reach}(a) \cap N_o$ *then* $\texttt{Granted}(a, n)$ *:=* $\texttt{True}$*.*
- *Else* $\texttt{Granted}(a, n)$ *:=* $\texttt{False}$*.*

Intuitively, this function refers to some policy configuration that grants access to n. If $\texttt{Granted}(a, n)$ = $\texttt{True}$, then there is a way for this actor to satisfy the access control policy for n (possibly under condition that he arrives at the right location).

Let us define a model state.

Definition 10 (State)**.** *A generated state for a model* $\mathcal{M}$ *is a tuple* $\langle \texttt{Reachable}(,), \texttt{Granted}(,) \rangle$*.*

Definition 11 ($\texttt{Accessible}(,)$)**.** *We define a boolean function* $\texttt{Accessible}(,) \subseteq N_a \times N$*:*

- $\texttt{Accessible}(a, n)$ *:=* $\texttt{Reachable}(a, n) \wedge \texttt{Granted}(a, n)$

Bootstrapping. Given a model $\mathcal{M}$= $\langle N, E \rangle$ produced by a modeller, the functions $\texttt{Reachable}(,)$, $\texttt{Granted}(,)$ and $\texttt{Accessible}(,)$ are initially computed from $\mathcal{M}$. First we compute a transitive closure of reachable locations:

- $\texttt{Reachable}(a, n)$:= $\texttt{Reachable}(a, n) \vee (\exists l \in N : \texttt{Accessible}(a, l) \wedge ((l, n) \in E \vee (n, l) \in E)$)

Notice that here we do not re-compute the function $\texttt{Granted}(,)$, and thus, eventually, the reachable objects set for each actor will increase only with locations that are not guarded by access control policy. Once $\texttt{Reachable}(,)$ is recomputed, it can be used to quickly evaluate whether an actor can reach certain locations in the original model (where may he end up).

Synthesis of AD-trees from bundles. We now discuss composition of generated attack-defence trees. An attack-defence tree $\texttt{ADT}(\eta, \alpha)$ is synthesised for a chosen attacker $\eta \in N_a$ and a target asset $\alpha \in N_o$. The root node is the bundle $access_\alpha$. For each leaf node of the type $access_b$ we can compute its value by referring to the corresponding AD bundle $\mathcal{B}_b$.

Bundle Value. In the simplest case we use propositional semantics for evaluating AD-bundles and, eventually, AD-trees (B. Kordy et al., 2012b). For leaf nodes of the type $access_n$, $access_n \equiv \texttt{Accessible}(\eta, n)$. For leaf nodes of the type $break_s$, $break_s \equiv \texttt{False}$ in the current synthesis approach. Thus, given a bundle for asset n, we can evaluate its value based on the values of the leaf nodes available. By updating the model state as attack progresses (more items become reachable to the attacker) we can evaluate the target bundle, once all its descendants become evaluated. As state changes monotonically, the process will eventually terminate.

5.2 Enriching attack trees with countermeasures

In this section we present a methodology developed in TRE$_S$PASS to automatically add countermeasures into an attack tree (Gadyatskaya, Harpes, et al., 2016). The method follows cost-benefit analysis, as it tries to identify the most suitable positions for countermeasures in the attack tree based on the reduction of attack probability. The process can be summarised as follows. An analyst starts the process by creating an attack tree, representing relevant threat scenarios, and then this attack tree is automatically decorated with countermeasure nodes, with suitable countermeasures taken from established libraries.

5.2.1 Background

The proposed approach builds upon a couple of concepts and methods, which we introduce next.

Cost-benefit analysis. Security risk treatment may be based on qualitative or quantitative analysis. When performing the latter, the analysts often proceed to a complex cost-benefit analysis, which help to select the appropriate countermeasures that will optimally reduce risks while having minimal security implementation costs. The *Return On Security Investment* (ROSI) is a commonly used concept that addresses both requirements (Bistarelli, Fioravanti, Peretti, & Santini, 2012; Harpes, Adelsbach, Peccia, & Zatti, 2007).

Libraries. Libraries are commonly used in risk assessment processes. They are a kind of knowledge bases, which contain risk information shared across multiple risk analyses. According to ISO/IEC 27001 (ISO/IEC, 2013a), risk treatment relies on lists of security controls, and the analysts are expected to estimate the residual risks after the implementation of such controls. Some libraries on cloud computing are starting to emerge, such as the Cloud Controls Matrix from the Cloud Security Alliance (Cloud Security Alliance (CSA), 2016), and the ISO/IEC 27017:2015 standard (ISO/IEC, 2015).

TRICK Service. TRICK Service (Tool for Risk management of an Information Security Management System based on a Central Knowledge base), developed by itrust consulting in Luxembourg, is a web-based risk assessment and management software tool for the identification, analysis and estimation of assets, threats, vulnerabilities, risk scenarios and security measures (itrust consulting s.à r.l., 2013). It helps the analyst to determine a list

of security measures to be implemented in order to reduce the impact or the likelihood of possible risk scenarios.

Risk analysis in TRICK starts with establishing the context by collecting information about the type and business processes of the organisation and filling in a table, according to ISO/IEC 27005:2011 (ISO/IEC, 2011). This information is used by the analyst to establish the most important assets considering the sector of the organisation. After the context definition, a brainstorming session identifies assets and risk scenarios in the organisation. Qualitative risk assessment is performed at this stage to allow the analyst to estimate the exposure to identified threats, vulnerabilities and risks. The next step consists in identifying the security measures that are already implemented in the organisation, and assessing their current implementation rate and cost, referring to norms, such as ISO/IEC 27002 (ISO/IEC, 2013b).

The analyst then estimates the *annual loss expectancy* (ALE) of each asset-scenario pair, by multiplying the impact (in euros) that a scenario could have, with the annual expected probability that a scenario could occur on the asset. A *risk reduction factor* (RRF) is associated to each asset-scenario-countermeasure triple. The RRF is a coefficient that expresses the negative influence of a security control on the ALE generated by the occurrence of a scenario on an asset. For a given security control in relation to a given scenario acting on an asset, its RRF is a value between 0 and 1, where RRF=0 means that the countermeasure is useless, and RRF=1 signifies perfect protection.

In order to ensure that the overall risk assessment, analysis and treatment process are correct, the analyst needs to come up with a (sufficiently) complete list of scenarios and evaluate their respective probabilities. If scenarios are too generic, it is very challenging to evaluate their probabilities (or occurrence rates). At the same time, for simpler attack steps, e.g. vulnerability exploitation, it might be easier to evaluate their chances to occur by relying, e.g. on the available statistics in the sector. To better estimate the residual ALE, we proposed to apply the attack tree formalism summarized.

Calculations with risk reduction factors. The risk reduction factor (RRF) is an attribute linked to a risk scenario. It is expressed as a percentage and corresponds to the value evaluated by the analyst (security expert or risk assessor), by which the success probability of the main attack will be decreased after all the applicable selected countermeasures are implemented in the information system.

5.2.2 Countermeasure selection

We consider that the analyst who is using TRICK Service can express threat scenarios as attack trees, and will perform the subsequent risk treatment steps using such trees.

The ROSI function. The Return On Security Investment (ROSI) function evaluates the investment made into security controls versus the obtained security improvement. The average yearly cost of implementing a set of new countermeasures M (denoted as **cost**(M))

corresponds to the investment, and the total ALE reduction obtained as a result of implementing these new countermeasures (denoted $\Delta ALE(M)$) corresponds to the yearly gains. Thus, for a set of controls M, $ROSI(M) = \Delta ALE(M) - \textbf{cost}(M)$.

Considering that **Risk** = **Impact**·**Probability** (Bistarelli et al., 2006), we set the difference in the annual loss expectancy $\Delta ALE(M)$ as the product of the **Impact** times the difference of yearly probability of occurrence without and with implementation of the set of countermeasures M (Roy, Kim, & Trivedi, 2012c).

The probability for the attacker to reach the goal and to implement the threat scenario can be evaluated through probabilities of atomic attack steps. At the same time, the impact of the attack tree, i.e. the impact in case the attacker reaches his/her goal and the threat scenario expressed in the attack tree has occurred, can be estimated independently from the tree. Thus we focus only on probability values and the selection of countermeasures based on how well they can reduce the attack success probability.

We consider that each countermeasure t has a possible effect on each attack node x. This effect is described by an **effectiveness** parameter, $\textbf{eff}(t, x) \in [0, 1]$, with $\textbf{eff}(t, x) = 0$ corresponding to a useless countermeasure for x, and $\textbf{eff}(t, x) = 1$ defining perfect protection against x. Effectiveness is defined so that the overall probability of the attack x mitigated by t, which we denote as x_t, is defined as $\textbf{Pr}(x_t) = \textbf{Pr}(x)(1 - \textbf{eff}(t, x))$. Thus, the higher the effectiveness parameter of the countermeasure in the given context, the lower the resulting success probability of the attack.

The step of evaluating the security posture by considering already implemented countermeasures in TRICK Service, can be directly executed on the attack tree. The analyst is expected to place the existing countermeasures as defence nodes in the attack tree. Computation of probabilities in presence of countermeasures and their effectiveness can be done via the standard bottom-up evaluation algorithm (B. Kordy, Mauw, Radomirović, & Schweitzer, 2010). As a result of this step, the analyst obtains an attack-defence tree adt, which evaluates the overall probability of the considered attack scenario as $\textbf{Pr}(adt)$.

New countermeasures from libraries. As we have mentioned, the de-facto standard for risk treatment is to use libraries of appropriate security mechanisms, such as (Bundesamt fur Sicherheit in der Informationstechnik, 2013; ISO/IEC, 2013b; NIST, 2013; PCI DSS, 2015; European Organization for Safety of Air Navigation, 2009). TRICK Service also implements a knowledge base of standard security controls compliant with norms like ISO/IEC 27002 (ISO/IEC, 2013b), amongst others.

In practice an organisation cannot implement all potential countermeasures, and often even implementation of the most critical security controls needs to be prioritised due to budget restrictions. Therefore, countermeasure selection needs to be guided by the cost-benefit analysis, in which we will consider costs of countermeasures versus their respective benefit (how well they can reduce attack probabilities).

We consider a customised library to be implemented as an association matrix **E**, where columns are known attacks, and rows are candidate security controls. For an attack x and a countermeasure t, the value $\textbf{E}[x, t] = \textbf{eff}(t, x)$, is the effectiveness of t to counteract the

attack x. If $\mathbf{E}[x,t] = 0$, then the countermeasure t is not applicable as a protection against x. Furthermore, each countermeasure t has a cost of implementation, denoted $\mathbf{cost}(t)$.

Assumptions on countermeasure selection. We assume each security control to be universal. It is applied once for the entire tree and it has effect only on a few nodes. Thus, for each attack node x and each countermeasure t such that t is applicable to x ($\mathbf{eff}(t,x) > 0$), we consider that t can be applied to x as a defence node everywhere it is applicable, at most once to avoid trivial solutions when cheap countermeasures are applied several times. The proposed approach is valid for the vast majority of practical countermeasures, except the ones when multiple defences of the same type can in fact improve protection (e.g. several security guards may be better than one, several locks on a door can be better than a single one).

This assumption does not preclude t from being selected as a countermeasure at another applicable attack node y, where it also reduces the success probability. We stress that the proposed solution works well for the cases when indeed separate security controls with the same name need to be introduced in different locations of the infrastructure. For instance, if there are two vulnerable doors that can be used by the attacker to get in, we will be able to propose two door locks as separate protection mechanisms.

These considerations imply that the total cost of each countermeasure is not affected by the number of times this countermeasure appears in the attack-defence tree.

5.2.3 Summary of the process

The different processes involved in this approach are described in more detail in the next.

Creating an attack tree. The analyst models the risk scenario and describes the attack in a pure attack tree at, e.g. with help of the ADTool (Gadyatskaya, Jhawar, et al., 2016; B. Kordy, Kordy, Mauw, & Schweitzer, 2013), estimates the success probability of each leaf node (all the atomic attacks), and computes the initial success probability of the main attack using the bottom-up evaluation process of ADTrees. The success probability of the root node of at can also be denoted as $\mathbf{Pr}(at)$. Let n be the number of attack nodes in at. The analyst also estimates the impact of the risk scenario with a real number value expressed in euros.

Identifying countermeasures. The analyst prepares the list of potentially applicable countermeasures from the libraries. Let m be the number of countermeasures in this list. For each countermeasure, the analyst estimates their security implementation cost. The security implementation costs of countermeasures (denoted as $\mathbf{cost}(CM_i)$ in Figure 5.1) are stored as real number values.

Estimating the effectiveness of countermeasures. With all the attacks coming from the initial ATree at, and the countermeasures coming from the libraries, ADTop generates an association matrix **E**. Based on the list of all the identified applicable countermeasures, the analyst estimates the $(m \times n)$ effectiveness matrix **E**, indicating the effectiveness of a

countermeasure i on an attack node j. We define $\mathbf{E}[i,j] = \mathbf{eff}(i\text{-th countermeasure}, j\text{-th attack})$.

Creating the full attack-defence tree. At this stage, all the applicable countermeasures selected in the association matrix **E** (with non-blank effectiveness values) are added to the initial ATree at. Thus the initial ADTree, adt, which contains all the applicable countermeasures is built. This step corresponds to linking all the applicable contermeasures to their corresponding attack nodes in the initial ATree as shown in Figure 5.1. Note that this representation is simplistic as one countermeasure is attached to each attack node.

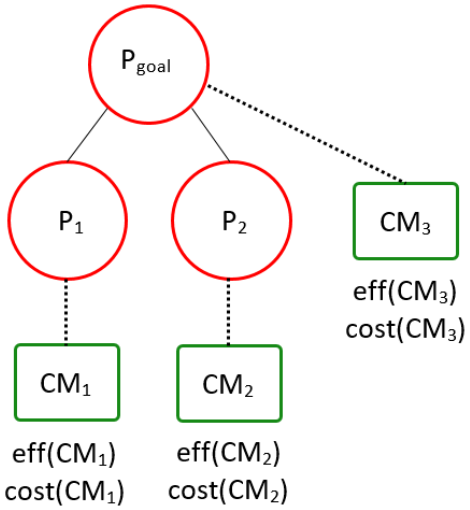


Figure 5.1: Basic ADTree.

Besides, several countermeasures may apply to the same single attack at a time. Thus, these countermeasures must be aggregated into a set of countermeasures (also called subgroup or meta-defence node), which includes all the selected countermeasures applicable for this given attack. Also, the same countermeasures may be selected several times, because they may be suitable to different attack nodes.

Solving the optimisation problem. Given all required input data, i.e. the targeted asset, attacks and countermeasures, effectiveness values and security implementation costs, the estimated impact (I_{goal}) of the risk scenario, and the initial success probability of the risk scenario (P_{goal}), we can define an optimisation problem, which consists in finding the countermeasures that have the best Return On Security Investment and minimise the following:

$$Min(I_{goal} \times P_{goal\ with\ M} + \sum_{i=1}^{n} x_i \times \mathbf{cost}(CM_i)) \tag{5.1}$$

where $\sum_{i=1}^{n} x_i \times \mathbf{cost}(CM_i)$ represents the sum of the security implementation costs of all the selected applicable countermeasures. x_i is a selection factor defined as follows:

$$x_i = \begin{cases} 1 & \text{if the countermeasure } CM_i \text{ is selected,} \\ 0 & \text{otherwise.} \end{cases}$$

The impact of the global attack, I_{goal}, multiplied by the success probability of the global attack without the set of countermeasures, $P_{goal\ without\ M}$ (equivalent to P_{goal}, P_{init} or **Pr**(at)) is a constant, thus not appearing in the above expression of our optimisation problem.

The residual success probability of a risk scenario for an optimal ADTree, $P_{goal\ with\ M}$ (equivalent to $P_{residual}$ or **Pr**(adt')), may differ from the one computed for a different optimal ADTree with another structure, so the specific structure of the ADTree which is being evaluated determines the formula to be solved for $P_{goal\ with\ M}$.

Added value of optimal countermeasure selection. The methodology presented in this section allows to compare the RRFs of the selected countermeasures in the optimal ADTrees in an equivalent manner to those implemented in TRICK Service. However, to compare them, the ADTrees should be distinctly designed and should include countermeasures separately, i.e. one per ADTree. Indeed, TRICK Service uses a linear model. The risk assessment tool considers the risk reduction for each individual countermeasure and sorts them in descending order, whereas in the optimal ADTrees, the RRF represents the risk reduction engendered by the implementation of all the selected optimal countermeasures. One of the advantages of ADTrees compared to linear risk assessment methodologies would be the greater level of detail resulting from the tree structure and from the effectiveness values, in contrast to the unstructured and independent risk scenarios and the more or less manually estimated RRF values. A risk scenario may be classified into more than one category, instead of confidentiality, integrity or availability only. The socio-technical attack built by the analyst may be a mix of all these categories and ADTop would still provide the analyst with some results, whereas it would be more complicated for TRICK Service. Indeed, to illustrate the same model in TRICK Service, the analyst needs to create different risk scenarios (one for each category) and make a mix of them using some ratios. Therefore, it is interesting to compare the different RRFs to see what changes on the non-linear optimisation model brings to the existing risk assessment methodologies.

Another aspect is that ADTop can provide the analyst with different optimal ADTrees for a specific risk scenario, based on a new approach for the estimation of the effectiveness values of countermeasures. The same analyst using the same expertise and knowledge bases and performing the estimation of the same countermeasures, implemented in the same organisational context, may obtain different optimal residual success probabilities and values of ROSI. The analyst is able to provide his/her client with a final overview of the information system, i.e. with the implementation of all the selected optimal countermeasures. In addition, the analyst is able to make various proposals, for instance depending on budget constraints for the implementation of the countermeasures, or on the maximum acceptable residual success probability for a given risk scenario.

Moreover, using custom attack and defence libraries, our model allows, for instance, to select only the countermeasures to be implemented for a specific risk scenario and their corresponding effectiveness values and security implementation costs. These libraries

constitute a knowledge base and help the analyst with the risk assessment process. They preserve gained knowledge from past risk analyses and can be shared among analysts, even non-experts.

5.3 Basics of Attack Defence Diagrams

We end this chapter by providing an intuitive discussion of the Attack Defence Diagram (ADD) formalism (Hermanns et al., 2016) using a running example. Recall that ADD, as attack-defence trees, can be used to model the 2-player game between attacker and defender. However, ADDs have been proposed fairly recently. Hence, it is worth looking at them a bit closer.

5.3.1 The ADD model

The central ingredients of ADDs are *events*. The most basic ones, *basic events* (BE), indicate an atomic, unique and significant happening in a real-world system. We distinguish *triggerable* and *non-triggerable* basic events. Triggerable BE causally depend on other events, but non-triggerable ones do not; triggers also appear in (Pietre-Cambacedes & Bouissou, 2010) to model instantaneous mode switches depending on a Boolean variable. We furthermore distinguish resettable and non-resettable basic events. The latter happen at most once, the earlier might happen several times, or even periodically.

Example 1. *As a running example, we consider a scenario where an attacker intends to get access to an email account of a company, so as to write embarrassing emails. In this setting, an attempt to guess the account password can be considered a non-triggerable BE, while the event BE of the attacker logging into the account is a triggerable BE, since it causally depends on obtaining the correct password.*

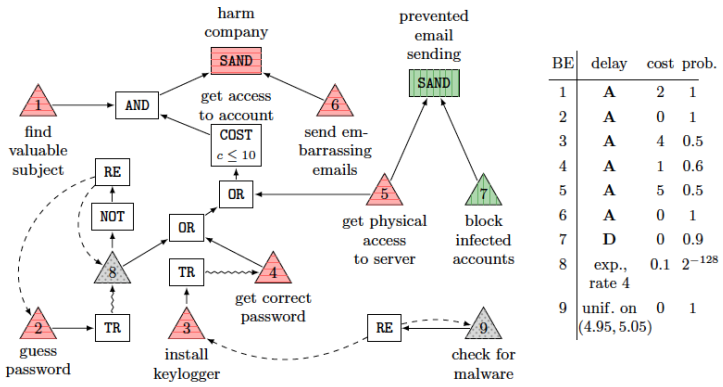
Sending an embarrassing mail can happen only once and cannot be undone after happening. Thus, this event is non-resettable, while checking for malware can be performed repetitively and thus is considered resettable.

In the above example, a brute-force random guess of a password has a certain probability of success, takes some time, and involves certain effort. To enable reasoning about such quantitative aspects within ADD, the BE can be decorated with *success probabilities*, with *time* durations, and with *costs* for delaying and executing single steps. The latter may right away represent money, or an abstract measure of skills the attacker or defender need to achieve or invest. The time point in which an event happens, or the time duration to achieve this goal (depending on the interpretation), is determined either randomly by sampling from some probability distribution (*time-driven* BE) or by one of the players (*player-driven* BE), either attacker or defender.

Example 2. *A single brute-force attempt of guessing a bit string of length* 128 *has a success probability of* 2^{-128}*. Situational conditions may dictate that every attempt causes (estimate) cost of* 0.1*, and takes on average* 0.25 *time units to effectuate. If only the average duration* T *is known, the provably best (and widely used) model for duration is the exponential distribution with rate* $1/T$*.*

We use three-valued logic (3VL) to represent the outcome of events. Basic events start with the truth value undefined (uu), and subsequently can execute successfully – changing to true (tt), or unsuccessfully – changing to false (ff).

Example 3. *The basic event representing the above attempt executes successfully with probability* 2^{-128} *and unsuccessfully with the remaining probability, after a delay sampled from an exponential distribution with rate* 4*.*



BE	delay	cost	prob.
1	A	2	1
2	A	0	1
3	A	4	0.5
4	A	1	0.6
5	A	5	0.5
6	A	0	1
7	D	0	0.9
8	exp., rate 4	0.1	2^{-128}
9	unif. on (4.95, 5.05)	0	1

Figure 5.2: An ADD representation of an attack on some email account

Apart from basic events, so-called composed events, or *gates*, form an integral part of an ADD. Gates enable the specification of complex situations, providing means to refine them into simpler events and eventually basic events.

Example 4. *Figure 5.2 displays an ADD for our running example. The solid edges form a directed acyclic graph where basic events are source vertices (triangular shaped) and gates are the remaining vertices (box shaped). If a solid edge points from* u *to* v*, we say that* u *is the input of gate* v*. Dashed and squiggle edges serve the purpose of resetting and triggering, as will be explained later. The colouring indicates whether the event is driven by or belongs to attacker (red with horizontal stripes), defender (green with vertical stripes), or is driven by time (gray with dots).*

Gates can roughly be considered as propagators of and operators on 3VL. Their value is ultimately determined by the truth values of basic events. The latter evolve in continuous-time, based on randomness as well as decisions of the players. The following gates are supported:

- *Logical gates* AND, OR, NOT, SWP serve as standard 3VL operators (where NOT swaps tt and ff and SWP swaps tt and uu). The latter is included for functional completeness, meaning that any 3VL operator is in fact supported;
- *Conditional gates* COST and IF propagate the input values ff and uu unchanged. When the input value becomes tt, the gates propagate tt given a side-condition is currently met, and ff, otherwise. This value is propagated as long as the input stays tt no matter how the value of the side-condition changes afterwards. The side-condition of a COST gate is whether one of the costs accumulated so far satisfies a specified cost bound. The side-condition of an IF gate is whether its second input, called *guard* has value tt at the moment.
- *Side-effect* gates TR and RESET are unary operators simply propagating the input value unchanged, thus representing logical identity. However, upon becoming tt, these gates cause a *side-effect*. The TR gate *triggers* basic events via squiggle edges, the RESET gate *resets* basic events via dashed edges.
- *Sequential gates* SAND and SOR extend the standard AND and OR by ordering their inputs sequentially in time. Whenever the first input of a SAND gate becomes tt, its second input is triggered, i.e. all basic events in its subgraph get triggered (apart from those having a squiggle edge from a TR gate within the subgraph). Similarly, whenever the first input of an SOR gate becomes ff, its second input is triggered.

Example 5. *We are now in the position to explain how the ADD in Figure 5.2 describes an attack on email accounts of a company. The goal of the attacker is to harm the company, represented by the red sink. By the* SAND *gate it boils down to* first *finding a juicy topic and getting access to some email account within the company and only* then *sending the embarrassing emails. The order in which the subgoals of* AND *are met does not matter.*

Thanks to the OR *gates, the subgoal* `get access to account` *can be achieved using three different approaches. All the basic events* 3, 4, 5, *and* 8 *are associated with positive execution costs representing the effort necessary to prepare the event. We assume the attacker has limited resources, modelled by the* COST *gate. It becomes satisfied if its input* OR *gate becomes satisfied and the accumulated cost at this moment is* ≤ 10*, i.e. attacker has not tried all possibilities so far.*

The computers in the company are nearly periodically `checked for malware`*. Therefore, the time-driven event* 9 *is executed every* ≈ 5 *time units. The periodicity is guaranteed by the* RESET *gate that resets the BE right after it executes. The* RESET *gate also implements the effect of the malware check – it removes the key-logger by resetting BE* 3*. Thanks to a* RESET *gate, also* `guessing the password` *may be repeated by the attacker arbitrary many times until it is successful. However, for every such guess, the execution cost is incurred.*

The goal of the defender is to prevent sending the email, represented by the green sink. This boils down to blocking all infected accounts before the embarrassing emails can be sent. The accounts can be blocked only after the company notices a suspicious activity in their building.

The players can decide to execute their active BE at arbitrary moments of time (except repetitively in zero time). These decisions of the players are based on the evolution of the truth values of all nodes in the ADD. For instance, a player may choose to execute a basic event at the time point when some gate turns $\mathtt{tt}$, at the time point when it turns $\mathtt{tt}$ for the third time, or exactly 2 time units after it has turned false $\mathtt{ff}$. The decisions may also use randomization, e.g. executing the basic event after a random delay (uniformly) from $[2, 3]$. Such a recipe when to drive basic events is called a *strategy* of the player.

5.3.2 Security metrics supported by ADDs

Metrics provide useful insight in the security level of the system under consideration, allowing security engineers to make design decisions, e.g., where to invest their security budget, or which security solution to implement. The ADD framework provides support for the analysis of various metrics, and their corresponding strategies.

What-if analysis. If we assume to know the strategy of both attacker and defender, i.e. we know exactly which player-driven BE they play and when, then we obtain a model that is fully stochastic, and we can calculate several security metrics. These metrics can involve all quantitative attributes, namely probability, time and cost. Typical examples for the model in Figure 5.2 include: *What is the probability of a successful attack?* asking about the probability that an embarrassing email gets sent, or conversely, that it is prevented: *What is the probability of a successful defence?* Due to the information contained in the model it is also straightforward to calculate cost metrics such as *What is the expected cost of a successful attack?* Apart from analyses that focus on the top level nodes, our metrics may involve any of the BE and gates in the ADD, such as BE 8 in Fig 5.2: *What is the probability of succeeding by correctly guessing the password? With what frequency are key-loggers removed from the system?* In our running example the latter corresponds to resetting BE 3 from $\mathtt{tt}$ to $\mathtt{uu}$. In models that capture the long-run situation with repetitive attack attempts, we can also ask questions such as:

What is the average number of attacks per year? How much is spent on average per year on defence? What is the expected ratio of defended attacks? Extending recent work on security metrics for attack trees (Kumar et al., 2015a) such metrics can be combined and represented as succinct diagrams showing different attack-defence scenarios, so as to show trade-offs among different attack strategies, for instance for the use of an enterprise risk manager to effectively plan defence strategies.

Game related questions. All these metrics can be similarly used in the following game questions:

- If we know the strategy of only one player, but not of its opponent, what is the *best response strategy* of the opponent for a given metrics. Suppose, in Figure 5.2 only the defender strategy is given: the defender blocks infected accounts right after the attacker gets physical access to the server. What is the counter-strategy to maximize the probability of a successful attack? And: what is the maximal probability?

- If we leave the strategies open for both players, what are their *optimal strategies*? An optimal strategy maximizes the interest of the player, assuming that the other player always plays a best-response strategy.

With formal preciseness, such questions are often computationally difficult (Brázdil, Krčál, Křetínský, Kučera, & Řehá, 2010), or even undecidable (Bouyer & Forejt, 2009). Nevertheless, we believe that heuristic or statistical approaches (David, Jensen, Larsen, Mikucionis, & Taankvist, 2015; Coulom, 2006) can provide useful results even for such complex models.

6 Generating and raking fraud scenarios from value models

Financial fraud is a very specific type of attack, which mostly target commercial electronic services, such as telecom services. We observe that these are organisations with their own peculiarity, commonly operating on top of a complex and highly-interconnected infrastructure. Moreover, providers of e-services operate in dynamic, highly competitive markets, which provides fertile ground for fraud. That is why dedicated models to analyse fraud in e-services have been developed, which are typically better suited than general purpose system models, such as the TRESPASS socio-technical model.

In this chapter, we describe the *e*^{3}*value* ontology, as well as our proposed fraud extension, which we call *e*^{3}*fraud*. The *e*^{3}*value* business value models is an alternative lightweight method for the identification and quantification of risks associated with e-service packages. In previous research, we have shown how the *e*^{3}*value* ontology (Gordijn & Akkermans, 2001) — with minimal extensions — can be used to model known telecommunication fraud scenarios (Ionita, Wieringa, Wolos, Gordijn, & Pieters, 2015) and proposed an automated approach to generate and rank such scenarios (Ionita, Wieringa, & Gordijn, 2016).

6.1 *e*^{3}*value* models

The *e*^{3}*value* modelling language was created – outside TRESPASS– to model an ideal business value model (Gordijn & Akkermans, 2003). Such a model shows what actors transfer in terms of economic value if all actors behave *honestly*. Figure 6.1 shows an *e*^{3}*value* model of a flat-rate[1] mobile phone subscription. We explain *e*^{3}*value* based on this running example.

The *e*^{3}*value* language supports the notion of *actors*. Actors are profit-and-loss responsible enterprises, non-profit organizations, or end-users. In this specific example, actors are telecommunication providers (provider A and B) and end users (user A and B).

The *e*^{3}*value* language also has the construct of *market segment*. A market segment groups actors that assign economic value to received or provided objects in the same way. For explanatory reasons, we do not use the notion of market segment yet.

[1] A flat rate, also referred to as a flat fee or a linear rate, is a pricing structure that charges a single fixed fee for a service, regardless of usage. (Gerpott, 2009)

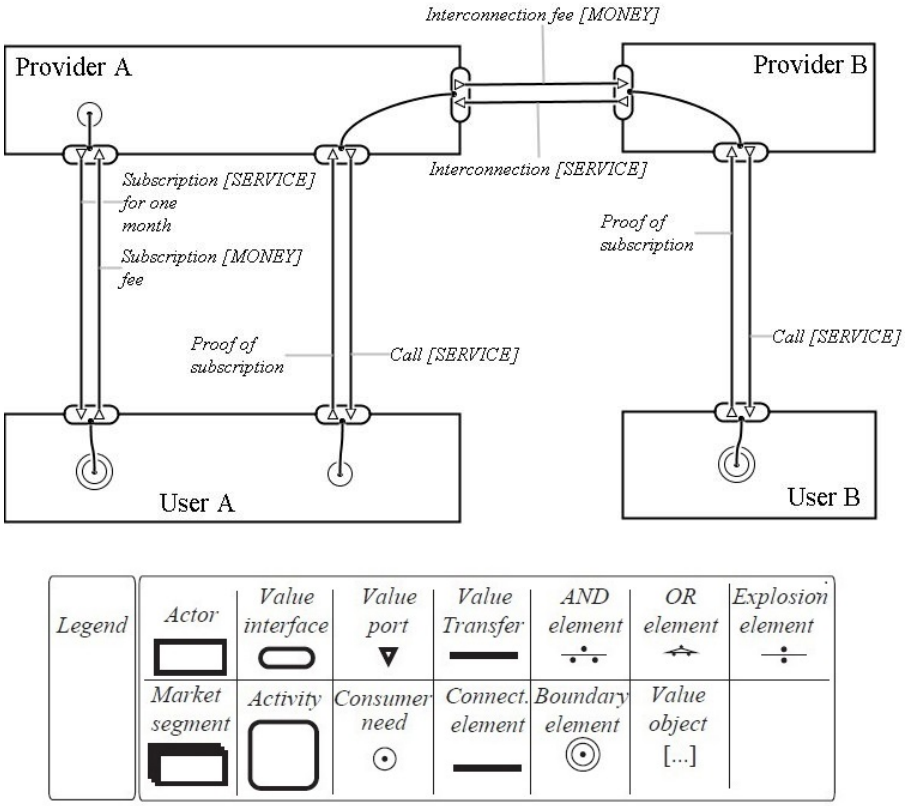


Figure 6.1: Ideal model: User A calls user B

Actors exchange things of economic value with each other, called *value objects*, via *value transfers* (visualized as straight line between actors). Value objects are physical objects or outcomes of services that are experienced by at least one actor in the model as of economic value.

In Figure 6.1 user A obtains a monthly flat-rate subscription (“Subscription for one month”) with Provider A and in return for this, the users pays Provider A an amount of money on a monthly basis (“Subscription fee”). The flat-rate subscription entitles the users to perform unlimited telephone calls to any other number. In return for presenting proof of this subscription to the provider, the provider delivers its service, which is a telephone call. In many cases, the caller and the callee do not have a subscription with the same provider, but rather with two providers, in Figure 6.1 providers A and B. So, to create a telephone connection initiated by user A to user B, provider A has to interconnect with provider B, since provider B is the operator user B has a subscription with. In other words, provider B delivers an interconnection service to provider A, and this service of value to provider A, because otherwise provider A could not create telephone connection outside its own network.

User B has his own contractual agreement with Provider B. However, this ideal model is built from the perspective of Provider A for whom the structure of this agreement is not

know nor observable and thus not represented. The only transaction between User B and Provider B which Provider A can observe is the telephone call.

An *e3value* model shows how actors do business which each other in a *contract period.* This is a period described by the contracts that describe the value transfers among actors shown in the diagram. An important property of an *e3value* model is *economic reciprocity*. Figure 6.1 shows various *value interface*s, containing *value port*s, transferring value objects (see the legend). The notion of value interface represents economic reciprocity, meaning that *all* value ports transfer objects of value, or *none at all*. For example, when provider A obtains interconnection from provider B, provider A will pay, as described by the contract, in the contract period. The same holds the other way around: If provider B is paid, provider B provides the interconnection service as described by the contract.

Finally, the *e3value* contains the notion of *dependency paths*. Such a path consists of *consumer needs*, value interfaces, value transfers *connection elements* (visualized as straight lines in the *interior* of an actor), and *boundary elements*[2]. A dependency path shows which transfers must happen, as a result of a consumer need. It does not show *when* they will happen, only *that* they will happen in the contract period described by the model. This is sufficient to estimate economic profitability in the contract period.

The technical and business processes by which these transactions are implemented contain a lot more detail and are not shown (Gordijn, Akkermans, & Van Vliet, 2000). It is even possible that the coordination processes that implement the commercial transactions implement a value transfer between actors A and B by means of a coordination process involving actors A, B and C. An *e3value* model abstracts from these operational details and shows commercial transactions only.

In Figure 6.1, user A needs to make a call to User B. In exchange for the call, User A pays a sum of money. By following the dependency path, we can see that provider A should obtain interconnection to provide the telephone call, and should pay for this interconnection. Finally, provider B delivers a telephone call service to user B.

For now, it is important to understand that in this *e3value* model *all* transfers on a dependency path are assumed to occur. In other words, the model shows what happens in reality, only all actors behave as agreed and expected. So, actors are always paying, and services are always provisioned. That is why we call such a model an *ideal* model; all actors operate honestly.

6.2 *e3fraud* models

In real-life, actors do not always behave as agreed and expected. They can perform intentionally or unintentionally in a different way. For example, an actor may not pay, or pay a wrong amount of money. Therefore, in our *e3fraud* extension, we consider two types of *intended* misbehaviours:

[2] A dependency path also may contain *AND*, *OR*, and *explosion/implosion* elements to represent dependency splits and joins, but for explanatory purposes, these are not used in Figure 6.1.

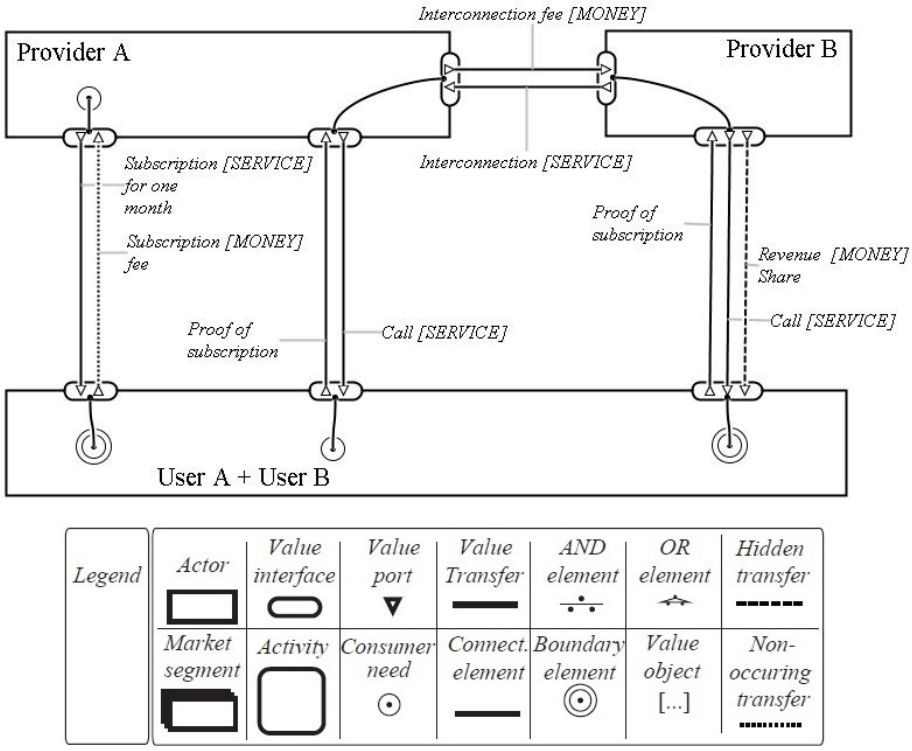


Figure 6.2: Sub-ideal model: User A calls himself and earns money

1. Transfers which should happen in the ideal model, do not happen at all in reality;
2. Transfers that happen in reality are not supposed to happen in the ideal model;

Furthermore, actors assumed to be independent in the ideal model may collude.

It is important to note that *e*^{3}*fraud* take the point of view of one actor, in this example provider A, who is the ToA (Target of Assessment). Next, we explain *e*^{3}*fraud* based on the same flat-rate mobile subscription.

Figure 6.2 shows an example of revenue sharing fraud, exhibiting both types of misbehaviours described above, as well as collusion. Rather than two end-users, as shown in Figure 6.1, we have now a single end-user A, who is exhibiting unwanted behaviour. This end-user has the same monthly subscription with Provider A as outlined in Figure 6.1. It is important to understand that this monthly subscription is based on a flat-fee tariff, which allows the user a to place a unlimited number of calls for free.

However, user A *also* has also access to a telephone hosted by provider B. The contract between user A and provider B states that for *received* call, user B gets part of the interconnection fee obtained by provider B (*Revenue Share)*. This is a common arrangement for 0900 numbers. Again, since we take the point of view of Provider A, we have no information on how User A obtained a contract with provider B or what the structure of their agreement is. For the fraud analysis, it is sufficient to assume such a bonus is being paid.

Furthermore, since the bonus pay-out is hidden to Provider A (the ToA), it is represented using a *dashed* line. Note that user A only uses provider B to *receive* calls.

To make matters worse, in this sub-ideal scenario we assume User A does not intend to pay his monthly fee to Provider A. As it is a non-occurring transfer with respect to the ideal model of the ToA, the *Subscription Fee* value transfer is represented using a *dotted* line.

There are several ways for user A to have a telephone served by provider B (e.g. a subscription), but we abstract way from the specific method, as for the fraud analysis, it is sufficient to model that user A pays a fee (e.g. a monthly fee), which is attributed to each call received by user A (since receiving calls in the only thing user A with provider B).

User A will now place as many calls as possible per month with provider B. As can be seen by following the dependency path, the *same* user A also terminates the call, but with his phone hosted by provider B. For each terminated call, user A receives a bonus. Considering that, in addition, he also intends to default on his payment of the Subscription fee, he is the position to make a a generous profit.

6.3 Automated generation of fraud scenarios

In (Ionita et al., 2016) we proposed an approach towards automatically generating fraud models, as described in Section 6.1, from value models, as described in Section 6.2.

Given a valid *e*^{3}*value* model, the associated tool[3] (further described in Sect 7.8 generates all combinations of possible deviations: *hidden transactions, non-occurring transactions and collusions*. These patterns were previously found to be the building blocks of several telecom fraud scenarios (Ionita, Koenen, & Wieringa, 2014). Furthermore, the user can select which of these transformations are applied and in which quantity. Each valid combination is then instantiated as new sub-ideal model. A sub-ideal model may contain any number of hidden transactions and non-occurring transactions but only one collusion. The number of actors colluding is configurable.

Hidden transactions are generated in three steps.

- First identify pairs of transactioning secondary (non-ToA) actors.
- Then, for each such pair, the profit/loss resulting from the dependency path of which this transaction is part, is computed for each actor.
- Finally, for the actor(s) with a positive result, a new outgoing transaction is added: this transaction takes a value of one third and two thirds of the positive result, respectively. The reasoning behind this is that if an Actor makes some profit on a dependency path, he might be be willing to pass on 1/3 or even 2/3 of that value to another actor, B, if that would motivate B to generate more traffic. This models an established practice in the services industry called Revenue Sharing (Ross, 2015).

[3] https://github.com/danionita/e3tools

The generation of hidden transactions is thus bound by the number of actors and transactions in the ideal model.

Non-occurring transactions are created by invalidating individual monetary transfers (that is, transfers marked as type $MONEY$). The restriction to monetary transfers is to limit state space explosion. The user may indicate that certain $MONEY$ transfers will always occur by marking them as as type $MONEY$-$SECURE$. This can be either because they are initiated by the provider itself, because safeguards are in place or simply to reduce the search space of sub-ideal models. This will prevent these transfers from being invalidated by the generation engine. The generation of hidden transactions is thus bound by the number of monetary transfers in the ideal model.

A Collusion takes place when two actors are acting as one: they pool their budgets and collectively bear all expenses and profit. By colluding, actors might deceive controls and invalidate expectations by appearing independent but in fact working together against the best interests of the provider. Therefore, only secondary actors (not the ToA) can collude. To generate collusions, pairs of secondary actors are merged into a single actor. The number of actors allowed to part of a colluding group is configurable. The generation of collusions is thus bound by the number of actors in the ideal model.

6.4 Ranking *e³fraud* fraud scenarios

Depending on the complexity of the initial ideal model, hundreds of even thousands of models might be generated. Many of these might not be possible due to existing controls or might be unlikely because they are not profitable for any of the actors.

To aid with selection and prioritization of risks, the tool provides several ways of ranking and grouping the set of generated models, described below. The prioritization is always carried out from the perspective of a single actor (the Target of Assessment), as described below.

In terms of value creation, non-ideal behavior causes a disruption in the financial result of the actors involved. This means that a non-ideal scenario can (1) cause a loss for the service provider and (2) trigger an unexpected gain for one of its customers or users. However, not all misuse is fraudulent (profitable) and not only fraud is damaging. As such, the software tool allows ranking based on Loss for the ToA, Gain for the secondary actors. Gain for a secondary actors is defined as the difference between the financial result of a that actor in the ideal case versus the sub-ideal case.

Ranking on Loss ranks risks on negative impact for the ToA (Loss) and profitability for the potential fraudster (Gain), while ranking on Gain uses the same two factors but ranks by Gain first. This is similar to the classical definition of risk as Impact times Likelihood of an event, except that we do not use Likelihood of the fraud but Gain to the fraudster. We use Gain to estimate the attractiveness of a fraud to a potential fraudster. A fraud with a higher gain for the fraudster is more attractive to the fraudster, and therefore more likely, than a fraud with a lower gain.

Furthermore, to allow for “what-if" analyses and easier navigation through the long list of sub-ideal models, results can be grouped based on who is colluding with who. Since each group is ranked independently, this allows investigating the most risky way each pair or group of actors can collude.

7 Overview of TRESPASS Quantitative Analysis Tools

In this chapter we provide consultant-level overviews of the analysis tools available in WP3. Note that, the full list of tools available in TRESPASS can be found in (The TRESPASS Project, D6.4.4, 2016).

7.1 Treemaker

Treemaker[1] is an approach to translate the TRESPASS socio-technical security model developed in WP1 into attack trees, that can be used as input for the analyses in WP3. As described in Section 3.1, Treemaker works by invalidating policies (Kammüller & Probst, 2013, 2014) to identify which locations to reach and which actions to perform, and generates attack trees based on these (Ivanova, Probst, Hansen, & Kammüller, 2015b, 2015d).

Input. The input for TreeMaker is a scenario that describes a model and a policy to invalidate. The policy can either be location-based (specifying that a certain asset is not allowed to reach a certain location) or action-based (specifying that a certain action is prohibited). Action-based policies usually take some asset as argument, to specify that the action is not permitted to be performed using these assets and having a certain role.

Output. TreeMaker generates attack trees as XML files in the language for describing attack-defence trees used by the ADTool as discussed in Section 7.2. Inner nodes in the generated trees have labels that indicate the actions performed by the sub-tree. Leaf nodes follow a grammar that enables their automatic parsing by tools such as the attack pattern library.

Status. TreeMaker was in Year 4 updated to exploit the physical, virtual, and social level to attack socio-technical models. On the physical level, attackers perform actions to obtain needed assets. On the virtual level, the required input for starting is identified, then the locations that the data in questions reaches, and finally on the physical level the attacker obtains the needed assets. On the social level, attackers perform social engineering attacks to make other actors obtain the needed assets.

Underlying method. The approach can be found in Section 3.1.

Example. The example is loosely based on the IPTV case study in TRESPASS. Figure 7.1 shows the example scenario, consisting of Alice's home, a bank with an ATM, and a bank

[1] https://trespass.itrust.lu/task/prepare/data/Treemaker

computer. Alice owns a card and a concomitant pin code to obtain money from an ATM, and a password to initiate transfers from her workstation via the bank computer. Some of the nodes are labelled with policies in dashed boxes; for example the money at the ATM requires a card with a pin code, as well as that very pin code in order to obtain money (modelled as input).

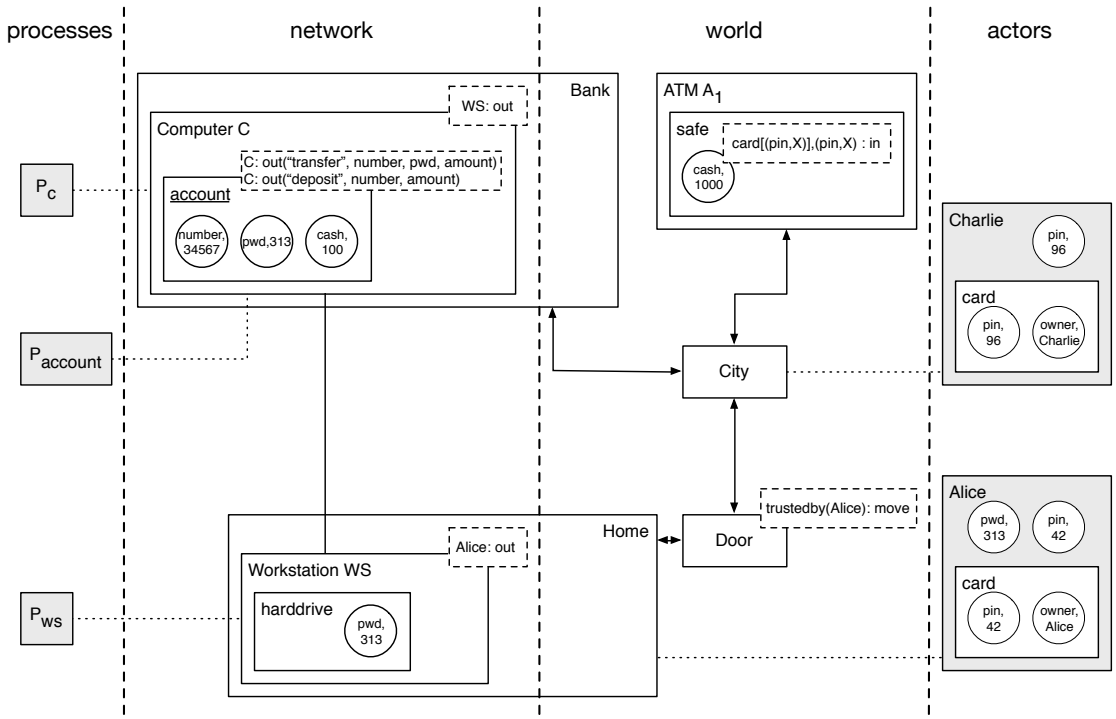


Figure 7.1: Graphical representation of the example system. The white rectangles represent locations or items, the gray rectangles represent processes and actors; actors contain the items or data owned by the actor. The round nodes represent data. Solid lines represent the physical connections between locations, and dotted lines represent the present location of actors and processes. The dashed rectangles in the upper right part of some nodes represent the policies assigned to these nodes.

Figure 7.1 shows a graphical representation of the model of our running example. The locations, represented by small rectangles, are connected through directed edges. Actors are represented as rectangles with a location, *e.g.*, Alice is at home and Charlie is in the city. Both actor nodes and location nodes can contain data and items represented as circles. In our example, Alice has a card that contains a pin code and Alice also has (knows) the pin code for her card. Actor nodes can also represent processes running on the corresponding locations. The processes at the workstation and the bank computer represent the required functionality for transferring money; they initiate transfers from Alice's home (P_{WS}), and check credentials for transfers (P_C).

We assume that the goal asset of the attacker is *cash*, which is available from locations *ATM A1* and *Computer*. Since TreeMaker does currently not support processes, we only consider the "physical" cash available at the ATM location.

Attack generation starts with the get *cash* action, which in turn will result in a get *cash* at *ATM* transformation. This results in a conjunction of going to the ATM, getting the credentials, and inputting the asset at that location, since the goal asset is directly contained in the ATM.

The credentials at the *cash* asset require a card with a matching pin. In the example system, both Charlie and Alice own matching assets, so attack generation results in two possible attacks, one using Charlie's card, another using Alice's card. Clearly, the first alternative does not necessarily represent an attack; generating such unwanted artefacts can either be prohibited by restricting permissible actors in the policy[2], or it can be dealt with in later phases that work on the generated attacks.

For the first possible attack, Charlie would use his own card and pin; this does not require further credentials. For the second possible attack, Charlie needs to obtain the pin and the card from Alice. Alice's location is *Home*, and to pass the path to this location, Charlie must fullfil the predicate *trustedby(Alice)*. This results in an action *social engineer Alice move Door*, which could in a later phase (such as the attack pattern library), be translated into a forceful entrance or pretending to be somebody who Alice trusts or is likely to let in her home. Once the location *Home* has been reached, Charlie has several options for obtaining the card and the pin:

- Social Engineer Alice to give him the card and the pin;
- Input card from Alice (stealing); and
- Input the pin from the card (skimming).

The generated attack takes account for all combinations hereof; some parts of the tree can be pruned or simplified in a later phase similar to (Vigo et al., 2014). Once the card and the pin have been obtained, Charlie moves to the location *ATM* and inputs the asset *cash*.

The resulting attack tree is shown in Figure 7.2. Not surprisingly, the transformation result contains identical sub-trees due to identical assets to obtain or identical patterns being transformed. Similar to the actions for obtaining items, these could be simplified by a follow-up pass.

7.2 ADTool

The ADTool[3] is a software for modelling attack-defense scenarios in ADTrees and ADTerms (an alternative term-based representation of ADTrees), and qualitative and quantitative security assessment with ADTrees; and as such the tool implements the theoretical foundations of the ADTree methodology. The tool supports ranking of attack scenarios based on quantitative attributes entered by the user, it is scriptable, and it incorporates attack trees with sequential conjunctive refinement.

[2] In this case, the owner of the card would not be allowed to be the actor performing the action.
[3] https://trespass.itrust.lu/jnlp/ADTool/ADTool.jnlp

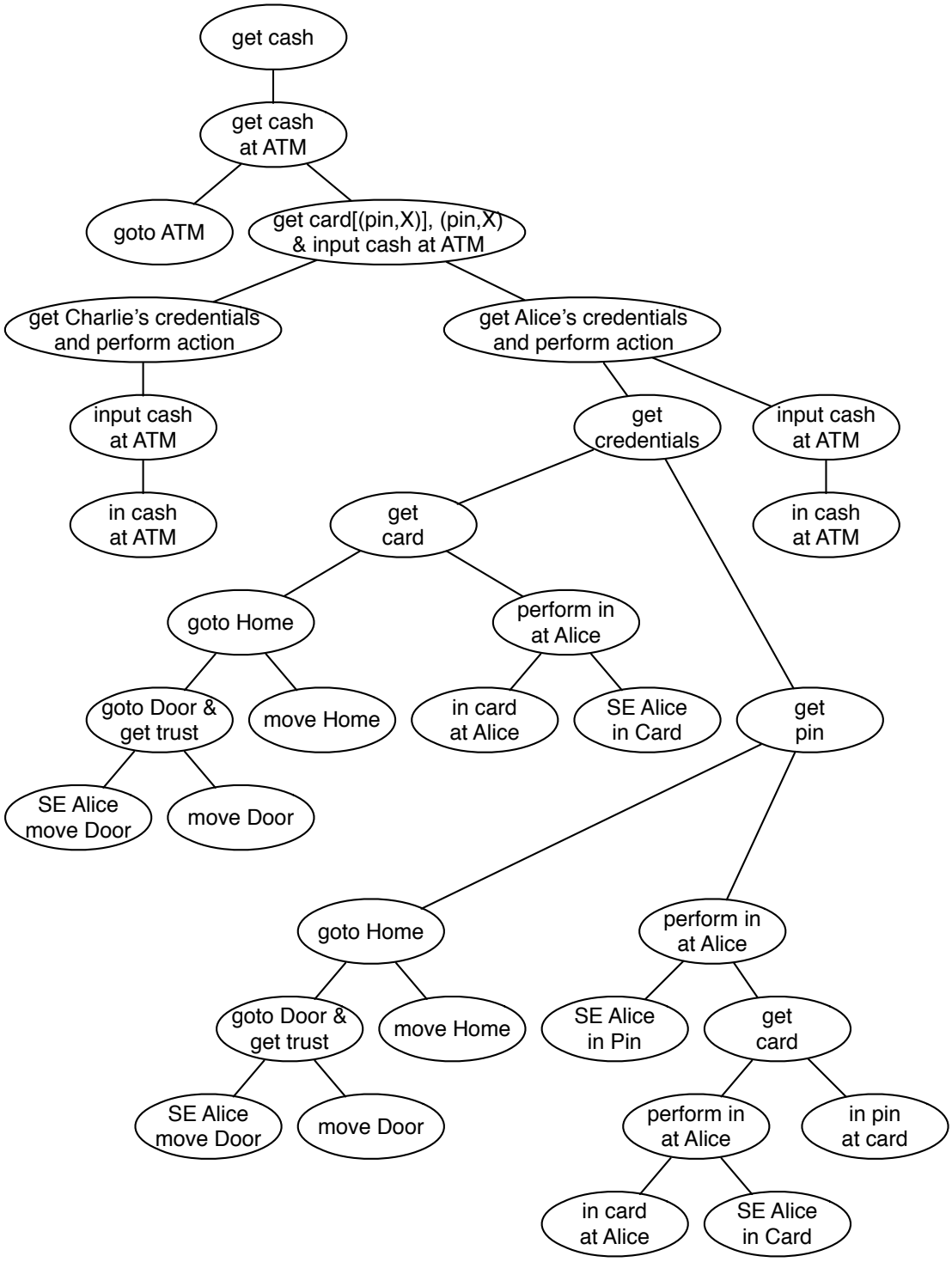


Figure 7.2: Result of attack generation for the example from Figure 7.1 using *cash* as the goal asset and Charlie as an attacker.

The tool and its capabilities are presented in more details in the tool demonstration paper (B. Kordy et al., 2013; Gadyatskaya, Jhawar, et al., 2016), (The TREsPASS Project, D3.1.1, 2013), and the tool manual (P. Kordy & Schweitzer, 2013).

ADTool is an open-source platform-independent software implemented in Java. The tool has a user-friendly GUI with several options for tree display, including zooming in and out and collapsing/expanding subtrees. The attack-defense scenarios can be edited in the tree form, as well as in the term form; the tool automatically maintains correspondence of representations after the user edits.

Input. The ADTool requires as input an attack-defence tree model (it can be a term-based model also), and assignment of attributes to the nodes. The tool is distributed as a standalone application (available a .jar or a web application), so there is no specific requirements on the system.

The language to describe attack-defence trees is based on ADTerms, that is, the term algebra for ADTrees. The syntax and description of this language can be found in (B. Kordy, Mauw, Radomirović, & Schweitzer, 2010). ADTool accepts as input files (typically with extension .adt) with attack-defence trees written in the ADTerm language.

Starting from Version 1.2, ADTool supports exporting and importing ADTrees written in the XML form. Representing ADTrees as XML files has been made possible by specifying an XSD schema describing the ADTree language, called *adtree.xsd*. The ADTool considers XML files conforming with this schema to be valid, and only such files can be imported by the application.

To perform quantitative analysis, the ADTool needs to receive an attack-defence tree (imported as an XML file in the XSD schema, or manually designed in the tool), which leaf nodes should be annotated with values from the chosen attribute domain.

Output. The ADTool contains several predefined attribute domains that cover the most common questions regarding an attack scenario, e.g, difficulty, minimal cost, minimal skill level, minimal time, and probability of success. For a detailed descriptions of these attribute domains we refer the reader to the ADTool manual. In total, the ADTool implements 13 predefined attribute domains.

Given an ADTree, an attribute domain, and a value for each basic action (leaf node), the tool computes automatically the values for all intermediate nodes. Naturally, this includes computing a value for the root node. The root node value answers the question related to the defined attribute domain.

Status. Recently, it was released the `ADTool2.0`, a new version of the ADTool with many new features, including ranking of critical attack scenarios, attack trees with the sequential AND (`SAND`) operator, and scriptability. The `ADTool2.0` is not a simple extension of the previous tool (B. Kordy et al., 2013), but a fully revamped, more advanced system. The `ADTool2.0` has been reimplemented using the advanced cross-platform Docking Frames library[4]. It is an open source software tool, freely available to the community[5].

[4] http://www.docking-frames.org/
[5] https://github.com/tahti/ADTool2

The ADTool2.0 includes many usability features, e.g., copy-paste of subtrees, handling of multiple trees, reorder of children nodes, and extended input format (automatically generated attack trees (Ivanova et al., 2015d) not conforming to the ADTool2.0 XML schema). The ADTool2.0 can handle and analyze *large trees* with several thousand nodes (automatically generated trees are typically of that size).

Further development of the ADTool will target analyses with multiple-attribute domains.

Underlying method. For analysis, the ADTool2.0implements the standard bottom-up approach in attack-defence trees (B. Kordy, Mauw, Radomirović, & Schweitzer, 2010). The tool perform ranking as described in Section 4.3.

Example. Let's consider a scenario where an attacker wants to steal money from a user's bank account. This can be accomplished in two ways: through an Automated Teller Machine (ATM) or by an online transaction. The authentication mechanism of ATMs typically requires a legitimate card plus a correct PIN code. A PIN code is a four-digit number which can be easily eavesdropped if the user is not cautious, or it could even be found in a backup note. The card instead, needs to be stolen by the attacker. The attacker also has the option to steal the user's credential and make an online transaction him self. A popular approach to do so is by phishing attacks or installing malware applications such as key loggers. However, this attack can be prevented by using an additional authentication factor, e.g, a key fob or a PIN pad.

The attack-defence tree model of the above scenario is depicted in Figure 7.3. This model has been created using the ADTool's graphical editor. The ADTool also allows the decoration of the tree with respect to several attribute domains. For this example, we choose the attribute *probability of success* and assign probability values to each basic action in the tree.

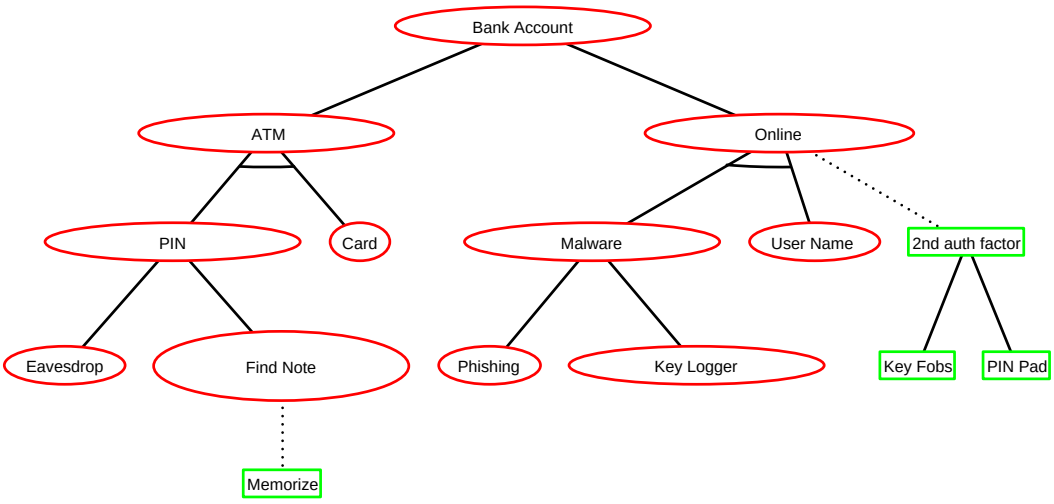


Figure 7.3: Attack-defense tree model.

The ADTool synthetises attribute values by using bottom-up approach. Figure 7.4 shows the attribute values assigned to each basic action as well as the intermediate values de-

termined by bottom-up approach. The results suggest that the probability of success of this type of attack is close to 0.1.

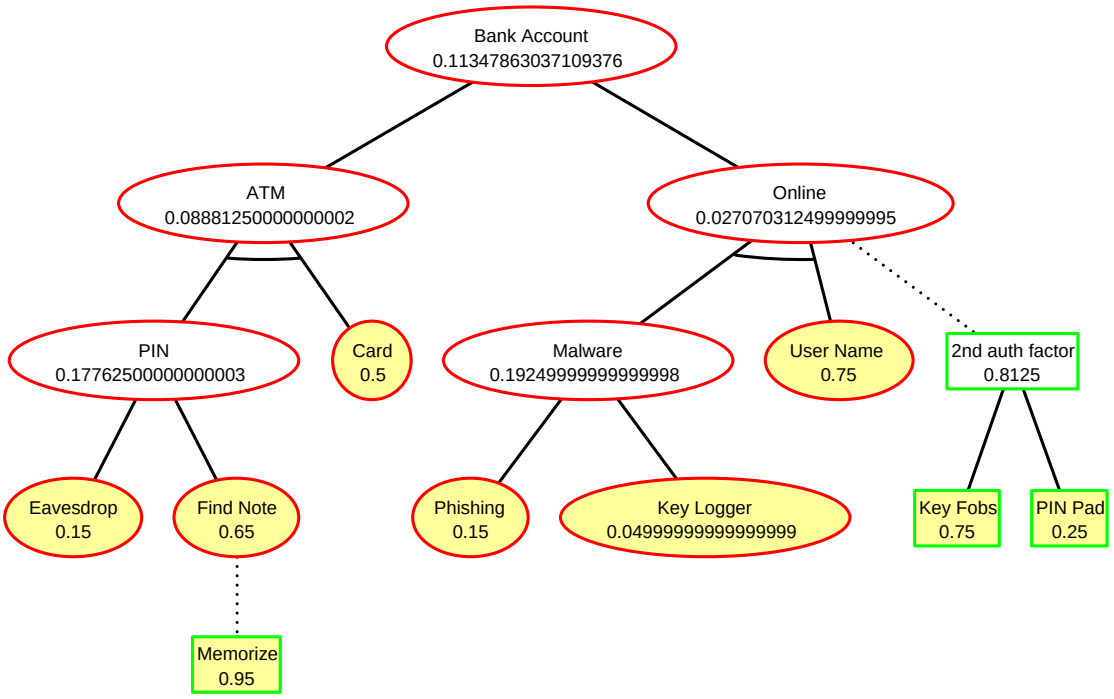


Figure 7.4: Attack-defense tree with attribute values.

The whole scenario, namely the attack-defence tree model together with the attribute values, can be exported to an XML file. Other supported output formats are pdf, latex, JPEG, or PNG formats.

7.3 Attack Tree Analyzer

The Attack Tree Analyzer tool[6] assesses if the considered infrastructure is secure against targeted rational profit-oriented attacks. The result of the analysis is an estimation for the upper bound 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. The tool contains several computational methods. Efficient bottom-up propagation rules are capable of coming up with the result in near-linear time, however they are limited to analyzing attack trees that have no common moves. For analyzing trees which have common moves, the two genetic approximation computational methods may be used.

Input. The inputs to the tool are the following:

[6] https://trespass.itrust.lu/task/prepare/data/Attack%20Tree%20Analyzer

- Attack tree model with annotated leaves
- Attacker profile (optional)
- Genetic algorithm profile (optional, required only when analysis is done using genetic algorithms)

The tool is distributed as a standalone application (available as a .jar archive).

The tool expects an XML-based attack tree description. XML files corresponding to the corresponding XSD schema are considered valid and can be analyzed.

Output. The output is typically the adversarial utility upper bound (security assessment is based on the value of adversarial utility upper bounds), and possibly some additional information. When the target enterprise is analyzed using the genetic algorithms, the output of the tool, apart from the utility estimation, contains the most profitable attack vector as well (if there is any).

Status. Currently the tool only considers an attacker model, and thus the entire satisfiability game may be represented in terms of a single player game played by the attacker.

Underlying method. The underlying model is called the Improved Failure-Free Model. For the description of the models as well as relevant computational methods the reader is referred to (Buldas & Lenin, 2013).

Example. An input to the tool is an attack tree as the one in http://research.cyber.ee/~antonc/ATA/Voting_Attack_Tree.xml. While the output is a profit value given XML format as shown next.

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

7.4 Attack Tree optimization (AT-top)

The attack tree optimization (AT-top)[7] is intended to perform multi-parameter optimization for attack trees. Thus, our analysis results into succinct diagrams to show trade-offs among different attack strategies, therefore enabling an enterprise risk manager to assess and prioritize defensive measures. Our tool can compute, e.g., the top-10 optimal attack paths for different quantitative attributes, e.g., the minimum time, budget, or skill level needed for the attacker (for a given attacker profile) to reach the goal, or induce the maximal damage.

Input. AT-top has three inputs:

[7] https://trespass.itrust.lu/task/prepare/data/ATtop

- An attack tree that describes how the single attack steps can be combined into an attack of the system under consideration. The attack tree can be generated from the socio-technical model by the Treemaker. Alternatively, the attack tree can be created manually with the ADTool, and then it can be automatically converted into the input format of AT-top.
- An attacker profile that assigns quantitative information (such as time, skill level required, etc) to each attack step.
- A security metrics that specifies the quantitative attributes to be computed. We plan to provide some standard options, potentially based on property specification patterns.

Output. The output of AT-top is the optimum attack values and path (minimum cost, minimum time, cost within minimum time). These can then be used to build Pareto frontier curves showing the trade-off between different optimum values. In addition to the attack values, At-top also can be used to get the attack path to reach these optimum attack values.

Status. AT-top is part of the TRE$_S$PASS tool chain available at:
`https://trespass-svn.itrust.lu/ATtop/`.

Underlying method. Technically, the tool is implemented via a translation to priced timed automata, which are then analysed by the model checker UPPAAL CORA.(The TRE$_S$-PASS Project, D3.3.2, 2015; Kumar, Ruijters, & Stoelinga, 2015b)

Example. Fig. 7.5 shows an attack tree taken from (Buldas, Laud, Priisalu, Saarepera, & Willemson, 2006). The example has been modified by adding sequential temporal dependencies. In this example, 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 some knowledge of computer security. This prerequisite though seemingly vague, is taken intentionally to show that our analysis tools can handle shared leaves/sub-trees. Thus, this atomic step – "Employ burglar with knowledge of computer security" can lead to two different attack paths modelled through a shared node. In one path the hired person finds a bug and exploits it to obtain the code via a network attack, and in another path he is physically involved in a burglary after being hired to steal the code. This dependency is again modelled through a SAND gate.

Our analysis take three inputs:

- An attack tree as shown in Fig. 7.5.
- An attacker profile and the associated attributes. We consider 2 different attacker profiles: a generic attacker and a software engineer. The associated values for the basic attack steps are given in Table 7.1.

- Security metrics :
  - Minimum time for the attacker.
  - Minimum cost for the attacker in US dollars.
  - Maximal cost to company in US dollars.

In Figure 7.7, we see that both the attack values and the choice of the 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 a 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 7.7, 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 7.6 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 the enterprise risk manager can use to effectively plan countermeasures.

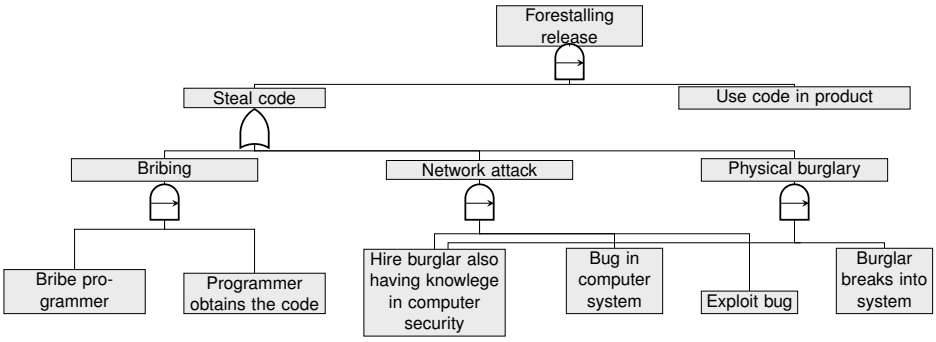


Figure 7.5: Example of attack tree for forestalling release of software

7.5 AT-Calc

Timed analysis of attacks is crucial to understand and predict the dynamics and nature of attacks. AT-Calc is a powerful tool for modelling and analysis of attack trees (ATs). It can efficiently model ATs and provide means to compute various dependability metrics,

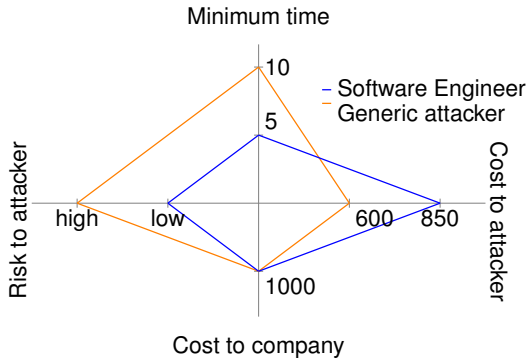


Figure 7.6: Optimal Resources(Time/Cost) for Generic Attacker/Software Engineer .

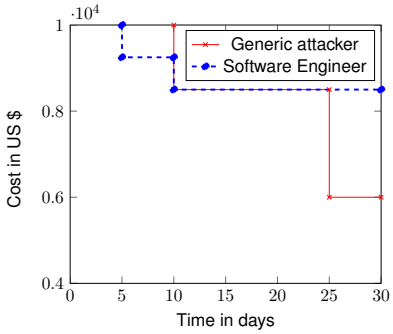


Figure 7.7: Pareto curve of attack tree in Figure 7.5

given attacks whose success probabilities are given by exponential and phase type distributions.

Input. The tool takes as input an attack tree and the values for the basic actions, such as probability of success, with a CDF (Cumulative distribution function) of the execution time. 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 the default choice. If there is no data available, experts can be invited to estimate t and the probability of success.

Output. Using the GUI at `http://fmt.ewi.utwente.nl/puptol/atcalc/`, a user can obtain various metrics:

- the probability of an attack for several attack times x, or
- the probability on an attack during an interval $[T1, T2]$, or

	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 7.1: Values used for annotating leaves of the attack tree as in figure 7.5

- the mean time for a successful attack.

The results can be given either in numbers, via the button *show result*, or as a plot, via the button *plot result*. The input and configuration of the web interface can be saved via the button *permalink.*

Status. The tool can be used by downloading a stand-alone version accessible via `git` from `http://fmt.cs.utwente.nl/tools/scm/dftcalc.git` in the branch `atcalc`, and via a web interface accessible at `http://fmt.ewi.utwente.nl/puptol/atcalc/`. AT-Calc is open source, but requires a license for CADP, which is free for academic institutions. The web interface is implemented with the use of PUPTOL and extends the downloadable version with a GUI, as well as the plot function.

Underlying method. AT-Calc exploits state-of-the-art model checkers and the highly compositional semantics of Input/Output interactive Markov chains (I/O-IMC) to enable the modelling and analysis of dynamic attack trees. (The TRESPASS Project, D3.3.2, 2015; Arnold, Guck, Kumar, & Stoelinga, 2015)

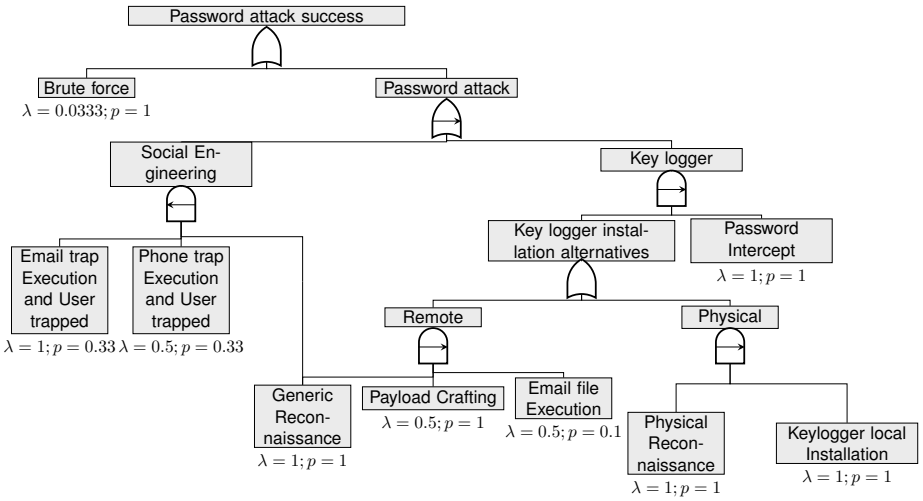


Figure 7.8: Dynamic Attack Tree model: attack on a password-protected file.

Example. Here, we have taken an example of an attack tree from the literature. The goal is to obtain the password of a password-protected file. There are two main ways to obtain it:

- by cracking it through brute-force, or
- by trying to obtain the password.

The input to the tool is an annotated attacked tree as shown in figure 7.8.

Figure 7.9 depicts the probability of a successful attack as a function over time. Our analysis results show that the attacker succeeds with probability of success around 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. 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 the sensitivity analysis is that we can figure out how much the output is vulnerable to input deviations.

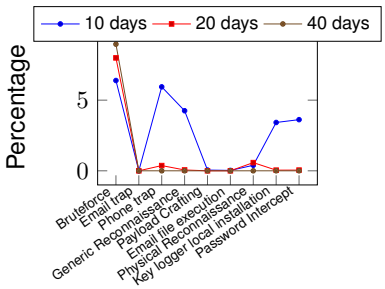


Figure 7.9: Sensitivity analysis for case: Password-protected file.

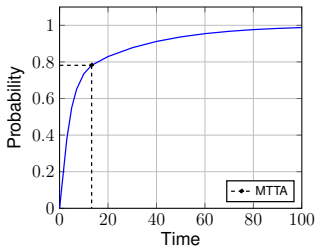


Figure 7.10: Probability of success for case: Password-protected file.

7.6 Attack Tree Evaluator

Attack Tree Evaluator[8] addresses multi-objective optimisation of attack trees. It can be used for analysing security properties of the model and computing the most promising attacks. The tool handles multiple conflicting parameters by computing the set of optimal solutions, defined in terms of Pareto efficiency.

Input. The tool takes as input an attack tree and the values for the basic actions such as probability of success and eventually their cost. The input attack tree for the tool should be in XML format that can be generated from the socio-technical model by other tools in the TRE$_S$PASS project, i.e., the Treemaker. The values for the basic actions are expected to be in the XML source.

Output. Depending on the values provided for the basic actions, the tool can compute different outputs. When only the probability values are provided for the basic actions, the tool computes the maximum success probability of an attack. When the probability and cost values are provided for basic actions, the tool computes the Pareto set of attacks with maximum success probability and minimum cost. In both cases, together with the solution the tool provides the set of corresponding basic actions. The tool can also compute the minimum cost of an attack, if the probability values are either 0, meaning the basic action is not performed, or 1, meaning the basic action is performed.

Status. The current version of the tool considers attack trees only.

[8] https://trespass.itrust.lu/jnlp/AttackTreeEvaluator/AttackTreeEvaluator.jnlp

basic action	probability of not performing action	probability of performing action	cost of not performing action	cost of performing action
Blackmail cardholder	0	0.2	0	30
Collect information	0.1	0.55	20	50
Threaten cardholder	0	0.3	0	30
Infiltrate premises	0.1	0.25	50	120
Impersonate trusted person	0	0.3	30	80
Impersonate TV technician	0	0.45	30	60
Impersonate technician	0	0.6	30	60
Jam remote-TV communications	0	0.65	35	100
Jam remote-card communications	0	0.65	35	100

Table 7.2: The values of probability and cost for the basic actions of the example.

Underlying method. For evaluating the security properties of an attack tree, the tool 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. For more details we refer the reader to (Aslanyan & Nielson, 2015), where similar method is described.

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 for fake services. A more detailed explanation of the example, as well as the attack tree, is presented in (The TRE$_S$PASS Project, D3.3.1, 2013).

Example input. The corresponding attack tree for the scenario is given in Figure 7.11. Possible probability of success and cost values for basic actions are listed in Table 7.2.

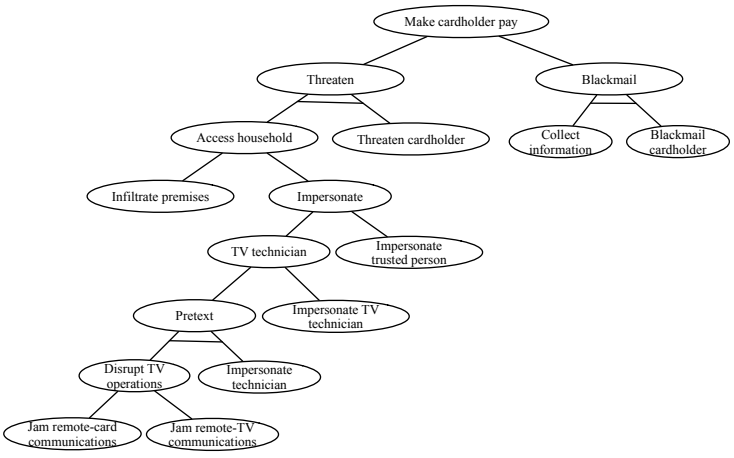


Figure 7.11: Attack tree for stealing money from the card-holder.

Example output. In order to detect the attacks with maximum probability of success and minimum cost, 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 7.12. 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.

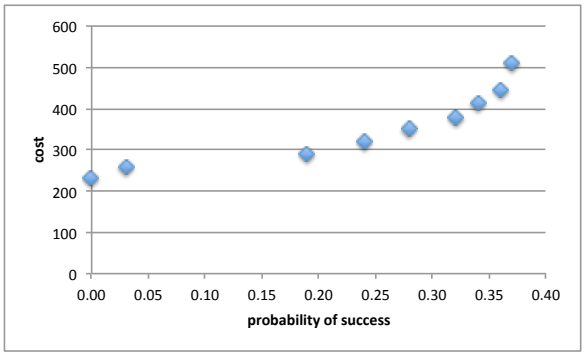


Figure 7.12: Pareto optimal solutions.

7.7 Statistical Model Checking

Statistical model checking (SMC) is a powerful, general technique that provides an alternative to models and analyses based on attack trees. It can be used to perform several quantitative analyses of system models, attacks, and countermeasures. Furthermore, the formalism underlying SMC (timed automata) is very expressive and has an intrinsic notion of time, making it relatively simple to incorporate time dependencies into both models and analyses. This has recently been exploited to show how attackers, in an *attack-defense trees*, can be given a fully stochastic timed semantics using timed automata including a notion of cost and probability of success for individual attackers actions. This then gives rise to an encoding of full attack-defense trees as a *network of timed automata* enabling automated analysis of the encoded attack-defense tree. By applying statistical model checking, full qualitative and quantitative analysis can be made of the attack-defense tree. In TRE_SPASS we use the UPPAAL SMC tool for statistical model checking.

Input. Input to UPPAAL SMC consists of a model specified as a network of timed automata. The model includes quantitative information, iuncluding stochastic parameters. Models can be created directly through the graphical user interface of UPPAAL SMC or by generating it (from some other tool) and importing the resulting XML file into UPPAAL SMC. As noted above, it is also possible to convert an attack-defense tree into a network of timed automata.

Output. UPPAAL SMC can perform a large number of analyses and (statistically) verify properties formulated in the built-in property specification logic. Furthermore, the graphical user interface allows users to explore the analysis results and generate graphs based on user defined parameters.

Status. UPPAAL SMC is based on the existing UPPAAL tool suite, and is continuously being improved and developed.

Underlying method. Statistical model checking works by evaluating and collecting data from a large number of runs of the model in question. Statistical analysis is then performed on the collected data, which is then used to compute the probability that the model satisfies the user-specified properties (within the given confidence interval). Thus, SMC can be seen as a combination of (traditional) model checking and testing/simulation. For a thorough introduction to statistical model checking and the underlying theory, we refer to the forthcoming UPPAAL SMC tutorial (David, Larsen, Legay, Mikucionis, & Poulsen, 2015).

Example. We illustrate the use of statistical model checking by a preliminary example, mainly used for exploring the potential use(s) of SMC in TRESPASS. The example is (loosely) based on the IPTV case, see The TRESPASS Project, D3.3.1 (2013) for more details on the case study.

Example input. The SMC model is composed of several (stochastic) timed automata. The two main components of the model are *attack steps* (shown in Figure 7.13) and *actors* (shown in Figure 7.14). The attack step modelling formalises individual steps that an attacker can take as well as the preconditions that must be met before a given attack step can be taken. This is similar in spirit to modelling attacks using attack trees, except that the full power of (stochastic) timed automata is available for modelling, e.g., making it trivial to incorporate notions of sequentiality and timing constraints. Modelling actors explicitly makes it possible to easily integrate victim and/or defender actions into the model, yielding more precise models and hence analyses.

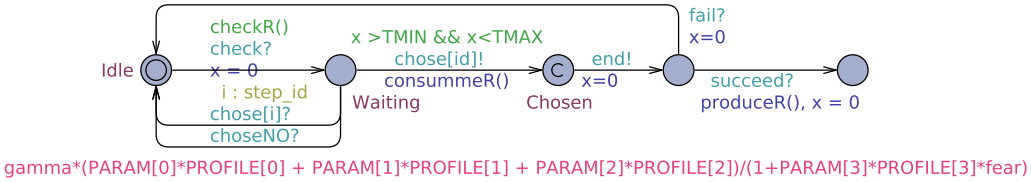


Figure 7.13: Modelling an attack step.

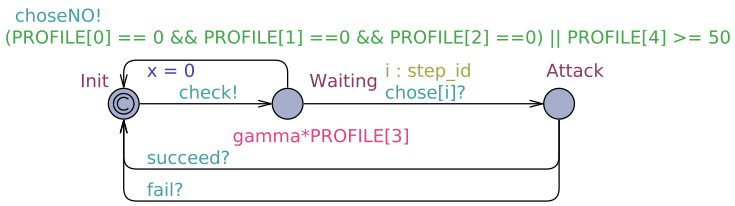


Figure 7.14: Modelling an actor.

Example output. Output from statistical model checking can take many forms, depending on the exact analyses performed: it is possible to focus on individual properties that can be (statistically) verified in the given model. An example of this is shown in Figure 7.15, where the highlighted "property" is actually a computation of the maximum (expected) exposure (to detection) for the attacker. Alternatively, it is possible to generate numerous kinds of graphs. In Figure 7.16 the expected value of different aspects of an attack, e.g., payoff and exposure, are plotted against time.

7.8 e3tool

e^3*tool*[9] is a merge between two existing projects:

1. The e3editor tool (developed outside of TRESPASS) with which users can construct and evaluate value models of networked business models:
   - Represent networked business models in terms of end users and enterprises, as well as the things of economic value they exchange with each other.
   - Assign economic value to the things exchanged, set pricing models, the number of customer needs, the actors involved and required investments.
   - Calculate the profitability of a networked business model for all actors involved, evaluate changes in participating actors, and increase of prices.
2. The e3fraud tool with which users can conduct of fraud assessment of a networked business model:
   - Automatically generate deviations from a given e3value model, in terms of (1) transactions that might not occur as agreed, (2) transactions that were not expected to occur and (3) collusion, where two or more actors thought to be independent act together against the interests of the provider.
   - Sort and filter these fraud scenarios based on gain for potential fraudsters or impact for the service/product provider.
   - Visualize the financial effects of fraud scenarios across a given range of occurrence rates.

By merging these two tools, users can now also manually construct (and analyse) e3fraud models.

Input. The *e*^{3}*tool* is a standalone tool so it does not take any input.

Output. The *e*^{3}*tool* can produce profitability charts from "normal" and fraud value models as well generate ranked lists of fraud scenarios and associated models.

Underlying method. The approach is described in Chapter 6.

[9] https://trespass.itrust.lu/jnlp/e3tools/e3tools.jnlp

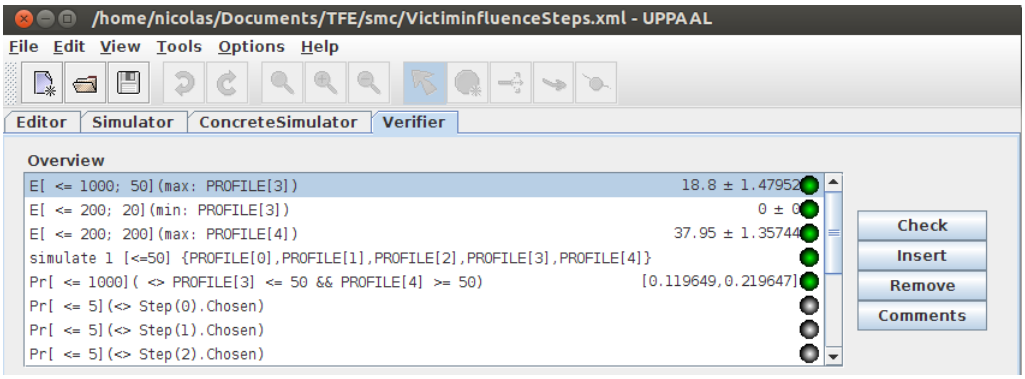


Figure 7.15: Statistical verification of a property using SMC.

Figure 7.16: Exploration of expected values.

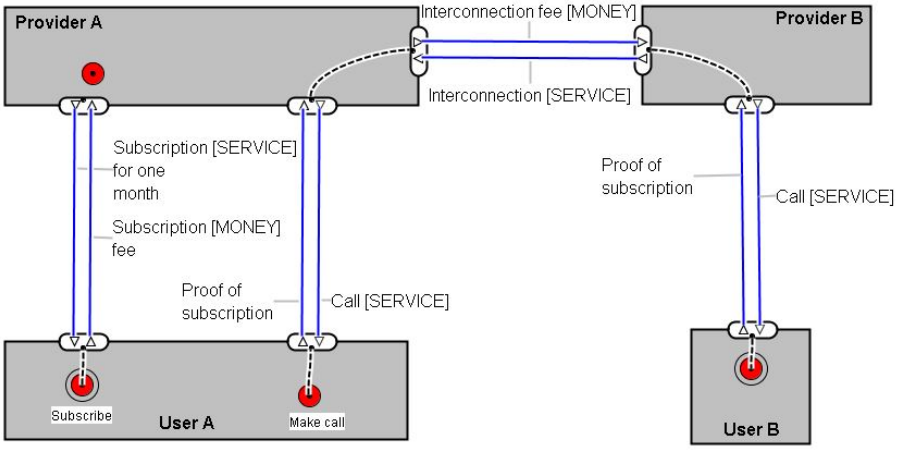
Tutorial: Conducting a fraud assessment of a value model

The tool contains several examples of e3value models. In this example, we will use the flat-rate service provision as a running example. A flat rate, also referred to as a flat fee or

a linear rate, is a pricing structure that charges a single fixed fee for a service, regardless of usage (https://en.wikipedia.org/wiki/Flat_rate).

A simple value model of a flat-rate telephony contract can be found under the "Examples" menu. A screenshot of this model is shown below.

Figure 7.17: A value model of a flat-rate mobile phone subscription
1- ideal mode - color.JPG



Step 1: Create an e3value model

An e3value model represents how actors exchange value (such as services, products or money) in an ideal world during a period of time called the contract period. For instance, the "flat-rate" example model represents the provision and payment of a fixed-price mobile telephony service during a period of one month. An e3value model assumes that all actors trust each other and all transactions occur as specified. The most important elements are:

- *Actors* are profit-loss responsible entities, such as organizations, customers and intermediaries. A *Market segment* is used to represent a group of similar actors (such as a pool of users). In the "flat-rate" example, “Provider A”, “Provider B”, “User A” and “User B” are actors.
- *Value transfers* are transfers of value objects, such as a payment or the delivery of a service. In the flat-rate example, all the lines between two actors are value transfers. Each transfer has an associated valuation, which can be edited via its context menu.
- *Value objects* are things of economic value which can be exchanged, such as money, services, products, knowledge or experiences. Value objects can be attached to value transfers via their context menu.
- *Dependency* paths are chains of economic transactions. In the flat rate example, there are two dependency paths: one for the subscription and one for the calling. Dependency paths do not represent processes. They merely indicate that in the

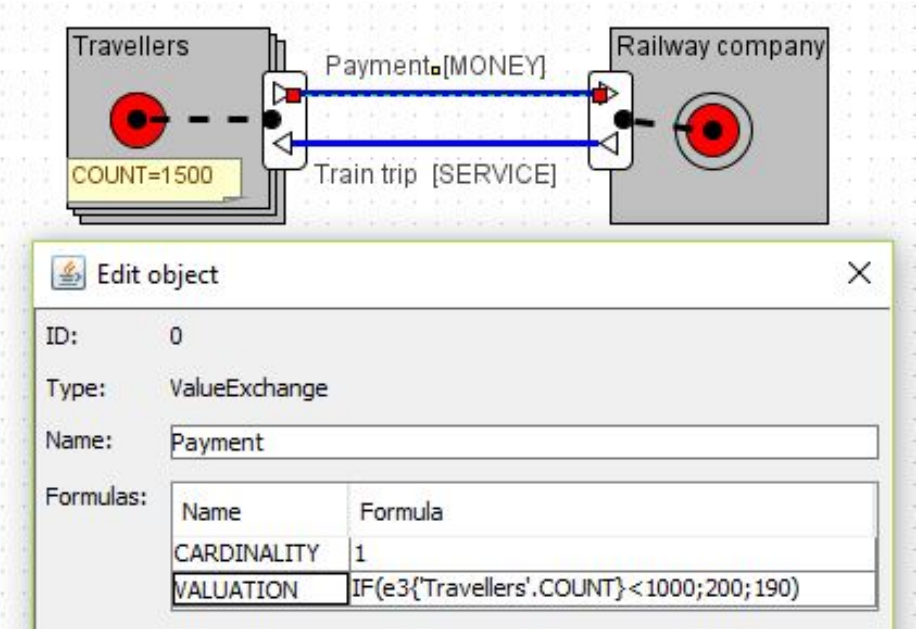
contract period, a consumer need triggers a certain combination of economic transactions, without saying when or how these transactions are performed.

- *Start stimuli* trigger a chain of economic transactions. Each start stimuli has an associated occurrence rate, which can be edited via its context menu. In the flat-rate example, “Purchase” and “Call” are such needs.

Attributes and expressions Most elements have several attributes accessible via their properties window, which can be opened by right-clicking individual elements. An attributes may take a numerical value (e.g. OCCURRENCES = 2), but can also be defined using logical Excel-like expression containing references to other attributes, either local or belonging to other objects in the model. References can be of four types:

- e3attributename -> legacy local
- e3#objectId.attributename -> legacy external (by uid)
- e3’objectName’.attributename -> new external (by name)
- e3ElementType(’name’).attributename -> old external (by name)

Figure 7.18: A value model of a flat-rate mobile phone subscription



Step 2: Generate and compare fraud scenarios

Clicking the "Fraud Generation" button in the Tools menu will open up a new window and automatically load the current model as a basis for fraud assessment. The fraud

assessment module is able to generate potential fraud scenarios based on this model and some fraud generation settings:

- The main actor represents the actor in the model whose point of view we are taking (I. E. The only trusted actor), also called the Target of Assessment or ToA. Selecting the main actor is needed to define the concept of hidden transactions, introduced below, and to assist with ranking the possible fraud models according to the potential loss for the ToA (further described in Section 2.3). In Figure 2, “Provider A” is the ToA.
- A need to be parameterized. This is the need whose occurence range will be used on the x-axis of the profitability charts. Furthermore, the average gain or loss across this range is used as the basis for sorting on gain or loss.
- A occurence rate interval for the selected need.

For this tutorial, launch the fraud generation module while having the "flat-rate" example open. Select "Provider A" as the main actor and the "Call" as the need to be parameterized. In the occurence range fields enter the estimated number of calls per month, e.g. 1-500. Finally, press the Generate button.

A list of generated scenarios appear in the center of the window. These scenarios are combinations of potentially fraudulent deviations: hidden transactions, non-occurring transactions and collusions. A sub-ideal model may contain any number of hidden transactions and non-occurring transactions but only one collusion. The number of actors colluding is configurable. For more details on how scenarios are generated refer to the in-depth description of the e3fraud module.

Step 3a: Sort, filter and group fraud scenarios

Fraud causes a disruption in the financial result of one or more the actors involved. This means that a fraud scenario should cause (1) a loss for the main actor and/or (2) a gain for some other actor.

As such, the tool allows **ranking and filtering** based on:

- *Loss*, defined as the negative difference between the financial result of the main actor in the ideal case versus the sub-ideal case
- *Gain*, defined as the pozitive difference between the financial result of any actor except the main actor in the ideal case versus the sub-ideal case

Furthermore, results can be **grouped** based on who is colluding with who. Since each group is ranked independently, this allows investigating the most risky way each pair or group of actors can collude. For more details on how scenarios are sorted refer to the in-depth description of the e3fraud module.

Step 3b: Visualize and analyse fraud scenarios

Clicking on a scenario in the list will display its profitability chart as well as a preview of its value model.

- The **profitability charts** (bottom-right) show how the evolution of the fraud and can be used to compare fraud scenarios with each other as well as with the ideal case, constructed in Step 1.
- The **preview** (bottom-left) shows how the fraud impacts the original value model:
  - Value transfers which do not take place and are marked using dashed lines. In the highest rated fraud scenarios of the flat rate example, the “Subscription Fee” transfer does not take place.
  - Hidden transfers occurring between secondary actors, not involving the ToA and are marked using dotted lines. In the highest rated fraud scenario of the flat-rate example, a “Revenue Share” is being paid out by Provider B to User A for each call received.
  - Colluding actors, which act as a single actor, pooling their budgets. This is represented using red highlighting. In the top rated fraud scenario of the flat-rate example, User A and User B are colluding. These visualizations can be used to quantitatively asses and compare the impact of the risk (in terms of loss for the ToA), the likelihood of the risk (in terms of gain for secondary actors) as well as the evolution of these factors across a given usage range.

Step 3c: Edit fraud scenarios

Double-clicking the preview of any fraud scenario’s will open it in the editor. Here, the fraud models of fraud scenarios can be customized, for instance by attaching additional fraud annotations as decribed below.

Attaching fraud annotations The following fraud annotations can be optionally added to a fraud model:

- *Value transfers* which do not take place can be marked by right-clicking and will show as dashed lines.
- Transfers can be marked as *hidden*, via the right-click menu, and will appear as dotted lines.
- Actors can be marked as *colluding*. They will then act as a single actor, pooling their budgets. This is represented using red highlighting. In the top rated fraud scenario of the flat-rate example, User A and User B are colluding.

8 Conclusions

In this deliverable we have overviewed the current activities in Task 3.4, and provided an overview of currently available analysis tools. We can conclude that Task 3.4 has achieved its goal, that is, providing a rich portfolio of tools and methods for the generation of attacks, preventive measures, and their ranking. The analysis tools available in the project are already quite mature, as they allow to analyse complex attack scenarios. They actually have been used and validated in documented case studies, such as (The TRESPASS Project, D7.4.2, 2016).

References

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

Arnold, F., Guck, D., Kumar, R., & Stoelinga, M. (2015). Sequential and parallel attack tree modelling. In *Computer safety, reliability, and security - SAFECOMP 2015 workshops, assure, decsos, isse, resa4ci, and sassur, delft, the netherlands, september 22, 2015, proceedings* (pp. 291–299). Retrieved from http://dx.doi.org/10.1007/978-3-319-24249-1_25 doi: 10.1007/978-3-319-24249-1_25

Aslanyan, Z., & Nielson, F. (2015). Pareto efficient solutions of attack-defence trees. In *POST* (Vol. 9036, pp. 95–114). Springer. Retrieved from http://dx.doi.org/10.1007/978-3-662-46666-7 doi: 10.1007/978-3-662-46666-7

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.

Behrmann, G., David, A., & Larsen, K. G. (2004). A tutorial on UPPAALI. In *Revised lectures of the international school on formal methods for the design of computer, communication and software systems (sfm-rt 2004)* (Vol. 3185, pp. 200–236). Springer.

Behrmann, G., Larsen, K. G., & Rasmussen, J. I. (2005). Optimal scheduling using priced timed automata. *SIGMETRICS Performance Evaluation Review*, *32*(4). doi: 10.1145/1059816.1059823

Bistarelli, S., Fioravanti, F., & Peretti, P. (2006). Defense trees for economic evaluation of security investments. In *Availability, reliability and security, 2006. ares 2006. the first international conference on* (pp. 8–pp).

Bistarelli, S., Fioravanti, F., Peretti, P., & Santini, F. (2012). Evaluation of complex security scenarios using defense trees and economic indexes. *Journal of Experimental & Theoretical Artificial Intelligence*, *24*(2), 161–192.

Bouyer, P., & Forejt, V. (2009). Reachability in stochastic timed games. In *ICALP* (pp. 103–114). Retrieved from http://dx.doi.org/10.1007/978-3-642-02930-1_9 doi: 10.1007/978-3-642-02930-1_9

Brázdil, T., Krčál, J., Křetínský, J., Kučera, A., & Řehá, V. (2010). Stochastic real-time games with qualitative timed automata objectives. In *CONCUR* (pp. 207–221). Retrieved from http://dx.doi.org/10.1007/978-3-642-15375-4_15 doi: 10.1007/978-3-642-15375-4_15

Buldas, A., Laud, P., Priisalu, J., Saarepera, M., & Willemson, J. (2006). 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., & Lenin, A. (2013). New Efficient Utility Upper Bounds for the Fully Adaptive Model of Attack Trees. In *Gamesec'13: Proceedings of the 4th conference on decision and game theory for security* (Vol. 8252, pp. 192–205). Springer.

Bundesamt fur Sicherheit in der Informationstechnik. (2013). *IT-Grundschutz-Catalogues, 13th version.*

Camtepe, S., & Yener, B. (2007). Modeling and Detection of complex Attacks. In *Proc. of Third Int'l Conference on Security and Privacy in Communications Networks* (pp. 234–243).

Cloud Security Alliance (CSA). (2016). *CCM V3.0.1., Jan. 21,* https://cloudsecurityalliance.org/download/cloud-controls-matrix-v3-0-1.

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and games (CG)* (pp. 72–83). Retrieved from http://dx.doi.org/10.1007/978-3-540-75538-8_7 doi: 10.1007/978-3-540-75538-8_7

David, A., Jensen, P. G., Larsen, K. G., Mikucionis, M., & Taankvist, J. H. (2015). Uppaal stratego. In *TACAS* (pp. 206–211). Retrieved from http://dx.doi.org/10.1007/978-3-662-46681-0_16 doi: 10.1007/978-3-662-46681-0_16

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

David, A., Larsen, K. G., Legay, A., Mikucionis, M., & Wang, Z. (2011). Time for statistical model checking of real-time systems. In *Proc. of computer aided verification (cav 2011)* (Vol. 6806, pp. 349–355). Springer Verlag.

European Organization for Safety of Air Navigation. (2009). *Threats, Pre-controls and post-controls catalogues.*

Forejt, V., Kwiatkowska, M., Norman, G., & Parker, D. (2011). Automated verification techniques for probabilistic systems. In *Formal methods for eternal networked software systems* (pp. 53–113). Springer.

Fraile, M., Ford, M., Gadyatskaya, O., Kumar, R., Stoelinga, M., & Trujillo-Rasua, R. (2016). Using attack-defense trees to analyze threats and countermeasures in an ATM: A case study. In *At the 9th IFIP WG 8.1 Working Conference on The Practice of Enterprise Modeling (PoEM) 8 - 10 November, 2016, Skövde, Sweden.* ((to appear))

Gadyatskaya, O. (2015). How to generate security cameras: Towards defence generation for socio-technical systems. In *Proc. of gramsec.* Springer.

Gadyatskaya, O., Harpes, C., Mauw, S., Muller, C., & Muller, S. (2016). Bridging two worlds: Reconciling practical risk assessment methodologies with theory of attack trees. In B. Kordy, M. Ekstedt, & S. D. Kim (Eds.), *Graphical models for security: Third international workshop, gramsec 2016, lisbon, portugal, june 27, 2016, revised selected papers* (pp. 80–93). Cham: Springer International Publishing. Retrieved from http://dx.doi.org/10.1007/978-3-319-46263-9_5 doi: 10.1007/978-3-319-46263-9_5

Gadyatskaya, O., Jhawar, R., Kordy, P., Lounis, K., Mauw, S., & Trujillo-Rasua, R. (2016). Attack trees for practical security assessment: Ranking of attack scenarios with adtool 2.0. In *International conference on quantitative evaluation of systems* (pp. 159–162).

Gerpott, T. J. (2009). Biased choice of a mobile telephony tariff type: Exploring usage

boundary perceptions as a cognitive cause in choosing between a use-based or a flat rate plan. *Telematics and Informatics*, *26*(2), 167-179. Retrieved from http://dblp.uni-trier.de/db/journals/tele/tele26.html#Gerpott09

Gordijn, J., & Akkermans, H. (2001, July). Designing and evaluating e-business models. *IEEE Intelligent Systems*, *16*(4), 11–17. Retrieved from http://dx.doi.org/10.1109/5254.941353 doi: 10.1109/5254.941353

Gordijn, J., & Akkermans, H. (2003). Value based requirements engineering: Exploring innovative e-commerce idea. *Requirements Engineering Journal*, *8*(2), 114-134.

Gordijn, J., Akkermans, H., & Van Vliet, H. (2000). Business modelling is not process modelling. In *Conceptual modeling for e-business and the web, ecomo 2000* (Vol. 1921, p. 40-51). Springer.

Hahn, E. M., Hartmanns, A., H., H., & Katoen, J.-P. (2013). A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods in System Design*, *43*(2), 191–232. Retrieved from http://dx.doi.org/10.1007/s10703-012-0167-z doi: 10.1007/s10703-012-0167-z

Harpes, C., Adelsbach, A., Peccia, N., & Zatti, S. (2007). *4th ESA International Workshop on Tracking, Telemetry and Command Systems for Space Applications, Darmstadt (D), 11–14/09/2007: Quantitative Risk Assessment with ISAMM on ESA's Data System.* Retrieved from https://www.itrust.lu/wp-content/uploads/2007/09/publications_TTC_2007_abstract_risk_assessment_with_ISAMM.pdf

Hermanns, H., Krämer, J., Krcál, J., & Stoelinga, M. (2016). The value of attack-defence diagrams. In *Principles of security and trust - 5th international conference, POST 2016, held as part of the european joint conferences on theory and practice of software, ETAPS 2016, eindhoven, the netherlands, april 2-8, 2016, proceedings* (pp. 163–185).

Ingols, K., Chu, M., Lippmann, R., Webster, S., & Boyer, S. (2009, Dec). Modeling modern network attacks and countermeasures using attack graphs. In *Annual conference on computer security applications acsac '09* (p. 117-126).

Ionita, D., Koenen, S. K., & Wieringa, R. J. (2014, October). *Modelling telecom fraud with e3value* (Technical Report No. TR-CTIT-14-11). Enschede: Centre for Telematics and Information Technology, University of Twente. http://eprints.eemcs.utwente.nl/25248/.

Ionita, D., Wieringa, R. J., & Gordijn, J. (2016, May). Automated identification and prioritization of business risks in e-service networks. In T. Borangiu, M. Dragoicea, & H. Novoa (Eds.), *Proceedings of the 7th international conference on exploring service science, iess 2016, bucharest, romania* (Vol. 247, pp. 547–560). London: Springer Verlag. http://eprints.eemcs.utwente.nl/27176/.

Ionita, D., Wieringa, R. J., Wolos, L., Gordijn, J., & Pieters, W. (2015, November). Using value models for business risk analysis in e-service networks. In J. Ralyté, S. España , & O. Pastor (Eds.), *8th ifip wg 8.1. working conference on the practice of enterprise modelling, poem 2015, valencia, spain* (Vol. 235, pp. 239–253). Berlin: Springer Verlag. http://eprints.eemcs.utwente.nl/26389/.

ISO/IEC. (2011). *27005:2011 Information technology – Security techniques – Information security risk management.*

ISO/IEC. (2013a). *27001:2013 Information technology – Security techniques – Information security management systems – Requirements.*

ISO/IEC. (2013b). *27002:2013 Information technology – Security techniques – Code of practice for information security controls.*

ISO/IEC. (2015). *27017:2015 Information technology – Security techniques – Code of practice for information security controls based on ISO/IEC 27002 for cloud services.*

itrust consulting s.à r.l. (2013). *TRICK Service,* https://www.itrust.lu/products/#products_licensed.

Ivanova, M. G., Probst, C. W., Hansen, R. R., & Kammüller, F. (2015a). Attack tree generation by policy invalidation. In R. N. Akram & S. Jajodia (Eds.), *Information security theory and practice - 9th IFIP WG 11.2 international conference, WISTP 2015 heraklion, crete, greece, august 24-25, 2015 proceedings* (Vol. 9311, pp. 249–259). Springer. Retrieved from http://dx.doi.org/10.1007/978-3-319-24018-3_16 doi: 10.1007/978-3-319-24018-3_16

Ivanova, M. G., Probst, C. W., Hansen, R. R., & Kammüller, F. (2015b). Attack tree generation by policy invalidation. In R. N. Akram & S. Jajodia (Eds.), *Proceedings of the 9th IFIP WG 11.2 international conference on information security theory and practice, WISTP 2015* (Vol. 9311, pp. 249–259). Springer.

Ivanova, M. G., Probst, C. W., Hansen, R. R., & Kammüller, F. (2015c). Transforming graphical system models to graphical attack models. In S. Mauw, B. Kordy, & S. Jajodia (Eds.), *Graphical models for security - second international workshop, gramsec 2015, verona, italy, july 13, 2015, revised selected papers* (Vol. 9390, pp. 82–96). Springer. Retrieved from http://dx.doi.org/10.1007/978-3-319-29968-6_6 doi: 10.1007/978-3-319-29968-6_6

Ivanova, M. G., Probst, C. W., Hansen, R. R., & Kammüller, F. (2015d). Transforming graphical system models to graphical attack models. In S. Mauw & B. Kordy (Eds.), *Proceedings of the 2nd international workshop on graphical models for security.* Springer.

Jhawar, R., Kordy, B., Mauw, S., Radomirovic, S., & Trujillo-Rasua, R. (2015). Attack trees with sequential conjunction. In *ICT systems security and privacy protection - 30th IFIP TC 11 international conference, SEC 2015, hamburg, germany, may 26-28, 2015, proceedings* (pp. 339–353).

Jürgenson, A. (2010). *Efficient Semantics of Parallel and Serial Models of Attack Trees.* Tallinn University of Technology, Faculty of Information Technology, Department of Informatics. Retrieved from http://digi.lib.ttu.ee/i/?496 (PhD Thesis)

Jürgenson, A., & Willemson, J. (2009). Serial model for attack tree computations. In D. Lee & S. Hong (Eds.), *Icisc* (Vol. 5984, p. 118-128). Springer.

Kammüller, F., & Probst, C. W. (2013). Invalidating policies using structural information. In *2nd International IEEE Workshop on Research on Insider Threats (WRIT'13).* IEEE. (Co-located with IEEE CS Security and Privacy 2013)

Kammüller, F., & Probst, C. W. (2014). Combining generated data models with formal invalidation for insider threat analysis. In *3rd International IEEE Workshop on Research on Insider Threats (WRIT'14).* IEEE. (Co-located with IEEE CS Security and Privacy 2014)

Kordy, B., Kordy, P., Mauw, S., & Schweitzer, P. (2013). ADTool: Security Analysis with Attack–Defense Trees. In K. R. Joshi, M. Siegle, M. Stoelinga, & P. R. D'Argenio (Eds.), *Qest'13* (Vol. 8054, p. 173-176). Springer.

Kordy, B., Mauw, S., Melissen, M., & Schweitzer, P. (2010). Attack-defense trees and two-player binary zero-sum extensive form games are equivalent. In *Proceedings of the first international conference on decision and game theory for security* (pp. 245–256). Berlin, Heidelberg: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=1947915.1947938

Kordy, B., Mauw, S., Radomirović, S., & Schweitzer, P. (2010). Foundations of attack-defense trees. In *Proceedings of the 7th international workshop on formal aspects of security and trust (fast)* (Vol. 6561, p. 80-95). Springer.

Kordy, B., Mauw, S., Radomirović, S., & Schweitzer, P. (2011). Foundations of attack-defense trees. In *Proceedings of the 7th international conference on formal aspects of security and trust* (pp. 80–95). Springer. Retrieved from http://dl.acm.org/citation.cfm?id=1964555.1964561

Kordy, B., Mauw, S., Radomirović, S., & Schweitzer, P. (2012a). Attack–Defense Trees. *Journal of Logic and Computation*, 1-33. (available online http://logcom.oxfordjournals.org/content/early/2012/06/21/logcom.exs029.short?rss=1) doi: 10.1093/logcom/exs029

Kordy, B., Mauw, S., Radomirović, S., & Schweitzer, P. (2012b). 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, P., & Schweitzer, P. (2013, October). The ADTool Manual [Computer software manual].

Kriaa, S., Bouissou, M., & Pièvre-Cambacédès, L. (2012). Modeling the stuxnet attack with BDMP: towards more formal risk assessments. In *7th international conference on risks and security of internet and systems, crisis 2012, cork, ireland, october 10-12, 2012* (pp. 1–8).

Kumar, R., Ruijters, E., & Stoelinga, M. (2015a). Quantitative attack tree analysis via priced timed automata. In *FORMATS* (Vol. 9268, pp. 156–171). Springer. Retrieved from http://dx.doi.org/10.1007/978-3-319-22975-1 doi: 10.1007/978-3-319-22975-1

Kumar, R., Ruijters, E., & Stoelinga, M. (2015b). Quantitative attack tree analysis via priced timed automata. In S. Sankaranarayanan & E. Vicario (Eds.), *Formal modeling and analysis of timed systems* (Vol. 9268, p. 156-171). Springer International Publishing. Retrieved from http://dx.doi.org/10.1007/978-3-319-22975-1_11 doi: 10.1007/978-3-319-22975-1_11

LeMay, E., Ford, M. D., Keefe, K., Sanders, W. H., & Muehrcke, C. (2011). Model-based security metrics using adversary view security evaluation (ADVISE). In *QEST* (pp. 191–200). Washington, DC, USA: IEEE. Retrieved from http://dx.doi.org/10.1109/QEST.2011.34 doi: 10.1109/QEST.2011.34

Lenzini, G., Mauw, S., & Ouchani, S. (2015). Security analysis of socio-technical physical systems. *Computers and Electrical Engineering*. Retrieved from http://www.sciencedirect.com/science/article/pii/S0045790615000671 (Available online 6 April 2015)

Lv, W.-P., & Li, W.-M. (2011, Nov). Space Based Information System Security Risk Evaluation Based on Improved Attack Trees. In *Proc.of 3rd Int'l Conference on Multimedia Information Networking and Security* (pp. 480–483).

Mateski, M., Trevino, C. M., Veitch, C. K., Michalski, J., Harris, J. M., Maruoka, S., & Frye,

J. (2012, March). *Cyber threat metrics* (Tech. Rep. No. SAND2012-2427). Sandia National Laboratories.

Mauw, S., & Oostdijk, M. (2005). Foundations of attack trees. In *Proceedings of the 8th international conference on information security and cryptology (icisc)* (Vol. 3935, p. 186-198). Springer.

Mehta, V., Bartzis, C., Zhu, H., Clarke, E., & Wing, J. (2006). Ranking attack graphs. In *Proc. of raid* (pp. 127–144). Springer-Verlag.

Moore, A. P., Ellison, R. J., & Linger, R. C. (2001). *Attack modeling for information security and survivability*. Carnegie Mellon University, Software Engineering Institute.

Niitsoo, M. (2010). Optimal Adversary Behavior for the Serial Model of Financial Attack Trees. In *Proc. of the 5th Int'l Conference on Advances in Information and Computer Security* (pp. 354–370).

NIST. (2013). *Special Publication 800-53 Revision 4. Security and privacy controls for federal information systems and organizations* http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf.

Othmane, L., Ranchal, R., Fernando, R., Bhargava, B. K., & Bodden, E. (2015). Incorporating attacker capabilities in risk estimation and mitigation. *Elsevier Computers & Security*, *51*, 41-61.

PCI DSS. (2015). *Payment card industry data security standards.*

Pièrre-Cambacédès, L., & Bouissou, M. (2010). Beyond Attack Trees: Dynamic Security Modeling with Boolean Logic Driven Markov Processes (BDMP). In *Proc. of european dependable computing conference* (pp. 199–208).

Pietre-Cambacedes, L., & Bouissou, M. (2010, April). Beyond attack trees: Dynamic security modeling with Boolean Logic Driven Markov Processes (BDMP). In *Dependable computing conference (edcc), 2010 european* (p. 199-208). doi: 10.1109/EDCC.2010.32

Ross, S. (2015). How does revenue sharing work in practice? *Investopedia*. Retrieved from http://www.investopedia.com/ask/answers/010915/how-does-revenue-sharing-work-practice.asp ([Online; accessed 12-December-2015])

Roy, A., Kim, D. S., & Trivedi, K. (2010a, March). Act: Attack countermeasure trees for information assurance analysis. In *Infocom ieee conference on computer communications workshops , 2010* (p. 1-2). doi: 10.1109/INFCOMW.2010.5466633

Roy, A., Kim, D. S., & Trivedi, K. S. (2010b). Cyber security analysis using attack countermeasure trees. In *Proceedings of the sixth annual workshop on cyber security and information intelligence research* (pp. 28:1–28:4). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1852666.1852698 doi: 10.1145/1852666.1852698

Roy, A., Kim, D. S., & Trivedi, K. S. (2012a, August). Attack countermeasure trees (act): Towards unifying the constructs of attack and defense trees. *Sec. and Commun. Netw.*, *5*(8), 929–943. Retrieved from http://dx.doi.org/10.1002/sec.299 doi: 10.1002/sec.299

Roy, A., Kim, D. S., & Trivedi, K. S. (2012b, August). Attack countermeasure trees (ACT): Towards unifying the constructs of attack and defense trees. *Sec. and Commun. Netw.*, *5*(8), 929–943. Retrieved from http://dx.doi.org/10.1002/sec.299 doi: 10.1002/sec.299

Roy, A., Kim, D. S., & Trivedi, K. S. (2012c). Scalable optimal countermeasure selection using implicit enumeration on attack countermeasure trees. In *Proceedings of the 42nd annual ieee/ifip international conference on dependable systems and networks* (p. 299-310). IEEE.

Schneier, B. (1999). Attack trees. *Dr. Dobb's Journal of Software Tools*, *24*, 21-29.

Schweitzer, P. (2013). *Attack-Defenses Trees,* http://satoss.uni.lu/members/phd-theses/pschweitzer13-thesis.pdf.

The TRE$_S$PASS Project, D1.2.2. (2015). *Final policy-specification language.* (Deliverable D1.2.2)

The TRE$_S$PASS Project, D1.3.4. (2016). *TRE$_S$PASS socio-technical security model and specification languages.* (Deliverable D1.3.4)

The TRE$_S$PASS Project, D2.4.1. (2016). *TRE$_S$PASS information system.* (Deliverable D2.4.1)

The TRE$_S$PASS Project, D3.1.1. (2013). *Initial requirements for quantitative analysis tools.* (Deliverable D3.1.1)

The TRE$_S$PASS Project, D3.1.2. (2015). *Final requirements for quantitative analysis tools.* (Deliverable D3.1.2)

The TRE$_S$PASS Project, D3.2.1. (2015). *TRE$_S$PASS extraction methods for stochastic models.* (Deliverable D3.2.1)

The TRE$_S$PASS Project, D3.3.1. (2013). *First report on stochastic analysis methods.* (Deliverable D3.3.1)

The TRE$_S$PASS Project, D3.3.2. (2015). *TRE$_S$PASS methods for stochastic analysis.* (Deliverable D3.3.2)

The TRE$_S$PASS Project, D3.4.1. (2014). *Attack generation from socio-technical security models.* (Deliverable D3.4.1)

The TRE$_S$PASS Project, D4.3.1. (2014). *Initial visualisations of socio-technical dimensions of information-security risks.* (Deliverable D4.3.1)

The TRE$_S$PASS Project, D5.4.2. (2016). *The integrated TRE$_S$PASS process.* (Deliverable D5.4.2)

The TRE$_S$PASS Project, D6.4.4. (2016). *The integrated TRE$_S$PASS tools.* (Deliverable D6.4.4)

The TRE$_S$PASS Project, D7.1.1. (2013). *Initial requirements for implementation of case studies.* (Deliverable D7.1.1)

The TRE$_S$PASS Project, D7.4.2. (2016). *Final report case study c.* (Deliverable D7.4.2)

Vesely, W., Goldberg, F., Roberts, N., & Haasl, D. (1981). *Fault tree handbook*. US Government Printing Office. (Systems and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission)

Viega, J., & McGraw, G. (2001). *Building secure software: How to avoid security problems the right way*. Addison Wesley Professional.

Vigo, R., Nielson, F., & Nielson, H. R. (2014). Automated generation of attack trees. In *Proceedings of the 27th computer security foundations symposium (csf)* (p. 337-350). IEEE.