predict
prioritise
prevent

TRESPASS

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

Deliverable D5.3.2

Best practices for model creation and sharing

Project:	TRESPASS
Project Number:	ICT-318003
Deliverable:	D5.3.2
Title:	Best practices for model creation and sharing
Version:	1.0
Confidentiality:	Public
Editor:	Wolter Pieters
Cont. Authors:	M. Davarynejad, M. Ford, D. Hadžiosmanović, O. Gadyatskaya, R.R. Hansen, D. Ionita, H. Jonkers, A. Lenin, Z. Lukszo, S. Mauw, B. Othman, W. Pieters, C.W. Probst, A. Tanner, R. Trujillo, J. van den Berg, J. Willemson
Date:	2015-10-30

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
CYB	Ben Othman	6
CYB	Aleksandr Lenin	6
CYB	Jan Willemson	1, 4, 5, 6, 8
UL	Olga Gadyatskaya	2, 7
UL	Rolando Trujillo, Sjouke Mauw	7
CHYP	Margaret Ford	2
IBM	Axel Tanner	4
BD	Henk Jonkers	3, 4
TUD	Wolter Pieters	all
TUD	Dina Hadžiosmanović	2, 5
TUD	Mohsen Davarynejad	2
TUD	Jan van den Berg, Zofia Lukszo	5
UT	Dan Ionita	3
DTU	Christian W. Probst	6, 4
AAU	René Rydhof Hansen	6
ITR		Contributions at a later stage

Quality assurance		
Role	Name	Date
Editor	Wolter Pieters	2015-10-28
Reviewer	Jeroen Barendse	2015-10-13
Reviewer	Sven Übelacker	2015-10-18
WP leader	Jan Willemson	2015-10-30
Coordinator	Pieter Hartel	2015-10-30

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

Acknowledgement: *The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318003 (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 v

Management Summary vi

1. Introduction 1
1.1. Motivation and challenges . . . 1
1.2. Goals . . . 2
1.3. Choices made . . . 3
1.3.1. Links with other deliverables . . . 3
1.4. Foreground and background . . . 5
1.5. Structure of the document . . . 5
1.6. Acknowledgements . . . 5

2. Sharing security information 7
2.1. Strategy . . . 7
2.2. Standards & protocols for security information sharing . . . 8
2.2.1. CybOX . . . 8
2.2.2. TAXII . . . 8
2.2.3. STIX . . . 8
2.2.4. OpenIOC . . . 9
2.2.5. MISP . . . 9
2.2.6. CIF . . . 9
2.3. NIST Guide to Cyber Threat Information Sharing . . . 9
2.3.1. Key Insights in Cyber Threat Information Sharing . . . 9
2.3.2. Relevance to the TRESPASS model sharing process . . . 12
2.4. Sharing attack patterns . . . 13
2.4.1. Attack pattern attributes . . . 14
2.5. Conclusions . . . 15

3. Model creation 16
3.1. Strategy . . . 16
3.2. Actor-network analysis . . . 16
3.2.1. Description of the method . . . 16
3.2.2. How the method has been used . . . 17
3.2.3. How the method could be used . . . 17
3.2.4. Why the method is a best practice . . . 17
3.3. ArgueSecure . . . 17
3.3.1. Description of the method . . . 17
3.3.2. How the method has been used . . . 18

3.3.3. How the method could be used . . . 18
3.3.4. Why the method is a best practice . . . 19
3.4. LEGO . . . 19
3.4.1. Description of the method . . . 19
3.4.2. How the method has been used . . . 19
3.4.3. How the method could be used . . . 20
3.4.4. Why the method is a best practice . . . 20
3.5. Other physical modelling . . . 20
3.5.1. Description of the method . . . 20
3.5.2. How the method has been used . . . 21
3.5.3. How the method could be used . . . 22
3.5.4. Why the method is a best practice . . . 22
3.6. ArchiMate and Architect . . . 22
3.6.1. Description of the method . . . 22
3.6.2. How the method has been used . . . 24
3.6.3. How the method could be used . . . 24
3.6.4. Why the method is a best practice . . . 25
3.7. The TRESPASS user interface . . . 27
3.7.1. Description of the method . . . 27
3.7.2. How the method has been used . . . 28
3.7.3. How the method could be used . . . 28
3.7.4. Why the method is a best practice . . . 29
3.8. Model development case: e-voting . . . 29
3.8.1. Description of the method . . . 29
3.8.2. How the method has been used . . . 30
3.8.3. How the method could be used . . . 30
3.8.4. Why the method is a best practice . . . 30
3.9. Implications for sharing . . . 30
3.10. Conclusions . . . 31

4. Model Pattern Library **33**
4.1. Strategy . . . 33
4.2. Reusing knowledge . . . 33
4.3. The library . . . 34
4.4. Conclusions . . . 35

5. Attacker Profile Library **36**
5.1. Strategy . . . 36
5.2. Agent-centric threat assessment . . . 36
5.2.1. TARA . . . 37
5.3. TRESPASS attacker profiles . . . 38
5.4. Including motivation . . . 40
5.5. The library . . . 41
5.6. Conclusions . . . 41

6. Attack Pattern Library **42**
6.1. Strategy . . . 42
6.2. Functions of the APL . . . 42
6.3. APL workflow . . . 44
6.3.1. Attack Expansion . . . 45
6.3.2. Attack Annotation . . . 47
6.4. The library . . . 47
6.5. Conclusions . . . 48

7. Attack Tree Sharing **49**
7.1. Strategy . . . 49
7.2. Open Attack Tree Library . . . 49
7.2.1. Requirements on Attack Tree Library . . . 49
7.2.2. Current Set Up . . . 50
7.2.3. Tree Sharing Format . . . 51
7.3. Synthesis of Attack-Defence Trees using a Library . . . 53
7.3.1. Syntactic Tree Definition Format . . . 54
7.3.2. Collection of ADTs . . . 57
7.3.3. Compliance . . . 57
7.3.4. Automated ADT extension . . . 59
7.4. Conclusions . . . 59

8. Conclusions **60**

References **61**

A. Project Summary **65**
A.1. Case Studies . . . 66
A.2. Overview of TRE$_S$PASS Integration . . . 67

B. Overview of existing approaches for MPL **70**
B.1. Unified Modeling Language (UML), UMLsec and CORAS . . . 70
B.2. Systems Modelling Language (SysML) . . . 71
B.3. Business Process Model and Notation (BPMN) . . . 72
B.4. ArchiMate . . . 73

Management Summary

Key takeaways

- Information sharing is essential in supporting model development practices;
- Innovative best practices for model development from TRESPASS are highlighted;
- Different libraries support model development and analysis: the Attacker Profile Library (APriL), the Model Pattern Library (MPL), and the Attack Pattern Library (APL);
- A new library is proposed for sharing entire attack trees.

This deliverable describes the progress made in Task 5.3 in months 13-36 of the TRESPASS project incremental to The TRESPASS Project, D5.3.1 (2013). Task 5.3 focuses on model creation, sharing and maintenance. The current deliverable describes creation and sharing of models, whereas model maintenance will be reported in The TRESPASS Project, D5.3.3 (2016).

In this deliverable, we first discuss the state of the art in sharing security information. We will then turn to the best practices created for development of TRESPASS models, and which shared data would be needed in such processes. In the remaining chapters, we describe three different TRESPASS libraries for sharing information that can be used in model creation: the attacker profile library, the model pattern library, and the attack pattern library. Finally, we discuss the sharing of entire attack trees.

In particular:

- We develop further the concept of attacker profiles and the associated library, and discuss their role in the overall computation framework;
- We instantiate the vision of a Model Pattern Library with four types of sharable objects (entire models, parts of models, templates, and elements) as the standard basis for modelling a large variety of real-world settings;
- We describe the architecture of the Attack Pattern Library (APL), including requirements for more extensive model annotations to make APL integration more flexible;
- We introduce the new Open Attack Tree Library.

1. Introduction

Appendix A provides the context for this deliverable in the TRESPASS project. It describes the overall summary of the project and the TRESPASS workflow. This introduction focuses on the goals of this deliverable, namely information sharing in the context of the attack navigator tools and processes.

1.1. Motivation and challenges

Cyber security breaches continue to increase along with the fast-growing adoption of IT technologies by small, medium, and large companies. A recent survey performed in the UK (*The 2013 Information Security Breaches Survey*, 2013) suggests that such security breaches cause losses in the order of billions of Euros per year. According to the same study, companies have on average raised their investment in security to 10% of their IT budget in 2013. The main drivers for information security expenditure include: preventing downtime and outages, complying with laws/regulations, protecting the reputation of the organisation, and protecting intellectual property.

Both the government and the private sector have recognised the need to protect against cyber security attacks, in particular, to tackle the security of critical and strategic infrastructure assets. A key factor in this respect is the gathering, analysis, and sharing of cyber security data, *e.g.* successful and unsuccessful attacks, known vulnerabilities and threats, attacker profiles and motivations.

In security and safety settings there is a need for information sharing, not only due to the need to better understanding the situation and possible type of risks and threats, but also to collect the data the trusted circle need to have in order to be able to draw a big picture of the situation. The need for sharing security information increases with the *volume* of the security data, its *velocity* and its *variety*. With the advent of new and sophisticated forms of attacks and *hacker industrialisation* (e.g. exploit-as-a-service, botnet-as-a-service, malware-as-a-service, etc.), information sharing is the first step towards making sense of data and a crucial pre-requirement for an active defence. Local system monitoring, accompanied by trusted forms of security information exchange and a proper system analysis facilitates global prevention of cyber attacks.

Cooperation and sharing have been acknowledged as a breakthrough in the fight against cyber attacks. Some examples can be found in Narasimhan, Varadarajan, and Chandrasekaran (2010); Frincke, Tobin, Mcconnell, Marconi, and Polla (1998); Cuppens and Miège (2002); Nojiri, Rowe, and Levitt (2003); Koutepas, Stamatelopoulos, and Maglaris

(2004) for Spam and DoS attacks detection and mitigation, and in Huang and Lee (2003); Locasto, Parekh, Keromytis, and Stolfo (2005); Gkantsidis and Rodriguez (2006) for securing peer-to-peer and *ad hoc* networks. Sharing security-related data allows the dissemination of system vulnerabilities, threats, and incidents within an integrated view. It also provides best security practices and solutions for continuous improvement of IT security products, which increases their commercial appeal.

However, the sensitive nature of information security data makes companies reluctant to share something so critical. Most cyber attacks have a discovery phase as a first step, and this type of information is very valuable for attackers who could identify weak points in the network, critical authentication servers that are not replicated, potential bottlenecks, machines infected with worms or other malware, service management boards in server, etc. This vital information is normally invisible to outsiders and it is unlikely that conducting scans from outside the network would reveal it. Consequently, sharing or publishing such sensitive information could potentially have devastating results for an organisation which is under attack.

Recognising both the benefits and the risks inherent in sharing information security data is the first step towards delivering more effective security solutions. Organisations face the challenge of defining what data to share, how, and to whom, in such a way that the economic profit is maximised. Several studies (Gal-Or & Ghose, 2005; Gao, Zhong, & Mei, 2013; Fang, Parameswaran, Zhao, & Whinston, 2012) have been published on these economic incentives in recent years, many of them based on game theory.

1.2. Goals

The modelling process used by today's risk assessment frameworks must support adaptivity of the models to a fast-changing threat environment. It has become impossible for one person to keep up with all these changes. Hence, the modelling process must provide a convenient way of distributing descriptions of new attacks and their properties to communities of people using risk assessment tools.

This issue was partly addressed by the FP7 project SHIELDS[1] that introduced a special online service called SVRS[2] for sharing model components, more specifically attack trees, describing various technical attacks. However, this service does not currently appear to be maintained or used on any scale and historically its focus has tended to be on issues relating specifically to software development.

Despite these issues, SVRS provides a useful prototype for the TRE$_S$PASS project to build upon. In particular, there is an opportunity to continue where the SHIELDS project left off, and implement a *standard library* of elementary attacks, to be integrated with the TRE$_S$PASS tools.

More specifically, the goals of Task 5.3 are the following:

[1] http://www.shields-project.eu/
[2] https://svrs.shields-project.eu/SVRS/

- Define technical specifications for the Model Pattern Library (MPL) and Attack Pattern Library (APL);
- Propose and develop best practices for sharing security information;
- Provide support for the risk assessment workflows, including sharing the models within different kinds of communities (in-house, inter-organisational, cross-border); and
- Provide technical specifications for model aggregation and integrity.

1.3. Choices made

Incentives for stakeholders, economic or otherwise, could play a key role in enabling the sharing of security data. If stakeholders benefit from sharing themselves, they will be more likely to cooperate. Although the TRE$_S$PASS project does not focus on incentives for sharing, it can contribute to the sharing landscape in a different way. By making it easy to share (parts of) security models, and making it easy to develop new models based on shared elements, we can provide the necessary infrastructure to enable sharing of information within the TRE$_S$PASS tools and processes.

To this end, this deliverable focuses on different ways to share modelling elements in TRE$_S$PASS. In particular, we will discuss libraries for the sharing of attacker profiles, model patterns, attack patterns, and attack trees. In addition, we will discuss best practices for model development and how sharing can contribute to those.

1.3.1. Links with other deliverables

The place of the elements covered in this deliverable in the project workflow is shown in Figure 1.1. For a general introduction into the project and its workflow, see Appendix A. Below, we discuss the relation with other deliverables.

1.3.1.1. D1.3.2: Extensibility of socio-technical security models

To enable the TRE$_S$PASS model to adapt to newly discovered aspects as well as to avoid bloating the model with rarely used features, the model offers extensibility as a feature. Extensions empower “Continuity through Extensibility”. The first use of extensibility is shown in The TRE$_S$PASS Project, D1.3.2 (2015) applying a geolocation extension to the ATM case study for generating risk heat maps (as tuple of latitude and longitude). A future extension is envisioned for cloud computing where geolocation can provide jurisdictional information. It can serve as a compliance extension to national and international regulations (e.g., data protection, Safe Harbour) and as a competitive element in product placement in particular (“Cloud Computing made in Europe”). If applicable, extensions for

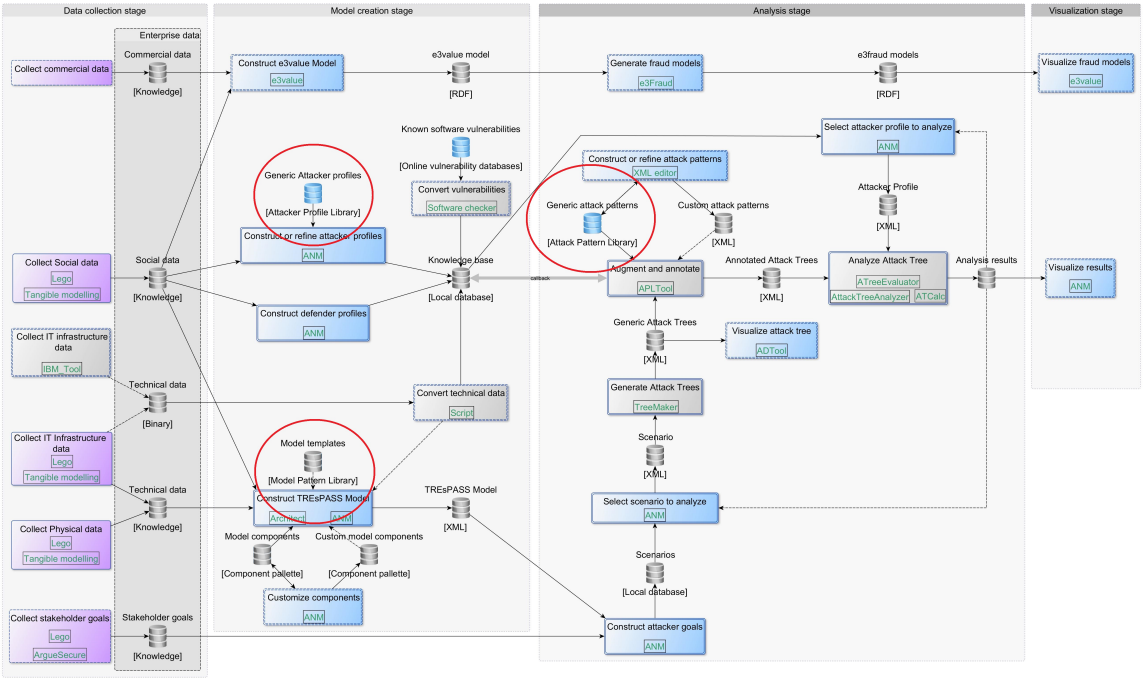

Figure 1.1.: An overview of the TRESPASS workflow. The red circles mark the parts covered in this deliverable.

similar scenarios can be subject of sharing, for instance geolocation information in ATM risk assessment and cloud computing.

1.3.1.2. D1.3.3: Dynamic features of socio-technical security models

Comparable to the extensibility deliverable (The TRESPASS Project, D1.3.2, 2015), dynamic feature in the TRESPASS model provide flexibility over time as described in The TRESPASS Project, D1.3.3 (2015). They can also contribute to model sharing by providing information about dynamics in the real world to other sharing stakeholders.

1.3.1.3. D5.4.1: First report on the integrated TRESPASS process

Sharing is part of the integrated processes described in The TRESPASS Project, D5.4.1 (2015).

1.3.1.4. D6.2.2: Final refinement of functional requirements

The use of the different types of libraries is represented in the tool integration diagram in The TRESPASS Project, D6.2.2 (2015).

1.3.1.5. D6.3.1: TRESPASS user interface

The TRESPASS user interface forms the project-specific platform for (digital) model development. In that sense, it forms a key part in the best practices for model development (Chapter 3). In addition, the user interface forms the gateway to the reusability of existing models and templates, termed the Model Pattern Library (Chapter 4).

1.4. Foreground and background

Chapter 2 contains an overview of background from third parties. ArchiMate and the Architect tool (section 3.7) are also considered background. Actor-network analysis (section 3.2) is 20% foreground and 80% sideground of third parties. TARA (chapter 5) is background from third parties. The rest of this deliverable contains foreground unless indicated otherwise.

1.5. Structure of the document

Chapter 2 discusses existing alternatives for security information sharing, and their relevance for TRESPASS. Chapter 3 provides an overview of best practices for TRESPASS model creation, and how they benefit from sharing. We then focus on three types of data sharing: model patterns, attacker profiles, and attack patterns. Chapter 4 studies approaches that can be used as a basis for the Model Pattern Library (MPL). Chapter 5 covers the TRESPASS approach to sharable attacker profiles and their integration in tree computations. Chapter 6 contains the description of the Attack Pattern Library (APL), and chapter 7 discusses the sharing of attack trees. Finally, chapter 8 draws conclusions.

1.6. Acknowledgements

Several contributions from research by master students, supervised by Dina Hadžiosmanović, Wolter Pieters, Jan Willemson and Sjouke Mauw, are included in this deliverable. This concerns:

- Rick van Holsteijn (Van Holsteijn, 2015): including attacker motivation in attacker profiles (chapter 5)
- Harikrishnan Pushpakumar (Pushpakumar, 2015): TARA method description (chapter 5)
- Ruud Verbij (Verbij, 2014): e-voting model development case study (chapter 3)
- Yiwen Zhu (Zhu, 2015): overview of attack pattern literature (chapter 2)

- Martin Popp Fredslund: synthesis of attack-defence trees from a library (chapter 7). We acknowledge also the contribution of Martin's co-supervisor, Dr. Ravi Jhawar.

2. Sharing security information

2.1. Strategy

Deliverable 5.3.1 (The TRESPASS Project, D5.3.1, 2013) discusses various scenarios of access to security information:

1. Sharing within an organisation,
2. Sharing between organisations,
3. Sharing in a cross-border environment, and
4. Publication.

All these scenarios assume that there is some party in the system (e.g., developer or end user) who is able to build the libraries to share. Even though one of the design principles of modelling and attack description languages is their simplicity, inherent complexity of describing real world artefacts and situations implies that we will not be able to assume most of the end users to be capable of creating the libraries. For example, it is safe to assume that not every organisation will have access to a modelling specialist.

Another concern with sharing security information is a fear of exposure. Even a seemingly harmless piece of data may turn out to be significant when augmented with some side information. Since at the release time it is impossible to foresee all the potential data inferences, organisations (and countries, being essentially instances of very large organisations for the purposes of our treatment) tend to be rather conservative in their data release policies.

All of the above implies that the library sharing ecosystem will have inherent asymmetry – there will be (many) more consumers of the information than there are providers. Hence, even with superior tool support, items 2 and 3 of the above list still have a high risk of failure simply due to information-economic considerations. Thus we will concentrate our attention and implementation efforts to the remaining items still viable in such an asymmetric setting.

In the first stage of implementation, basic technical support for the first scenario will be developed. It will have a simple file-based architecture, where the libraries will initially be distributed packaged together with the main application toolset. Updating the libraries may be accomplished by updating the corresponding files in the installation either manually or using a packaging tool.

In the second stage, the fourth scenario (publishing) will be implemented. As the simplest technical solution, a file-based distribution mechanism will be supported; more integrated level of sharing will become possible once the whole toolset matures into a unified platform.

In the remaining of this chapter, we will review existing sharing initiatives as a basis for library development in TRE$_S$PASS. In the following chapters, we will then look into TRE$_S$PASS-specific sharing needs.

2.2. Standards & protocols for security information sharing

A secure and trusted information-sharing protocol is a key factor to enable participants to share information easily. There are numerous options for security data-sharing.

2.2.1. CybOX

Cyber Observable eXpression: is a scheme for representing, characterising and communicating high-fidelity information about cyber events. Most common security information including network traffic properties, OS, disk, files, and more can be expressed in CybOX. CybOX provides the possibility to express events that take place during an attack, such as the deletion of a file. Also it has the possibility of describe states and properties of IT assets to perform incident response and event management. CybOX also handles contextual information intended to represent the behaviours of interest of the participants in a cyber security context.

2.2.2. TAXII

Trusted Automated Exchange of Indicator Information: TAXII enables threat information sharing between trusted entities that could vary from IP addresses to discovered vulnerabilities and preventive/defensive courses of action to be coordinated.

2.2.3. STIX

Structured Threat Information Expression: STIX is an extensible XML-based language that conveys potential cyber threat information. The TAXII cyber threat information binding being exchanged between subscribers and consumers is represented in the XML-based STIX schema. The type of data exchanged includes cyber observables, incidents, exploits, courses of action as well as cyber threat actors.

2.2.4. OpenIOC

Open Indicators of Compromise: OpenIOC is an open source XML schema that describes indicators and technical characteristics of known threats, attack methodologies, and common OS information that point to a compromise. It overlaps with other standards like CybOX that attempt to represent similar system characteristics that could indicate compromise.

2.2.5. MISP

The Malware Information Sharing Platform (MISP) is originally developed in Belgium and is an information sharing platform to share technical characteristics of malware within a trusted community without having to share details of an attack.

2.2.6. CIF

The Collective Intelligence Framework (CIF) is a cyber threat intelligence management system.

2.3. NIST Guide to Cyber Threat Information Sharing

In this section we summarise the current draft of the NIST Guide to Cyber Threat Information Sharing and outline the useful ideas that can be learnt from this document (Johnson, Badger, & Waltermire, 2014).

2.3.1. Key Insights in Cyber Threat Information Sharing

- *Lifecycle of information*: all organisations participating in the information sharing process shall be able to manage both the information they publish and the information they receive throughout the full lifecycle (from creation or acquisition to disposal).
- *Many benefits of information sharing but many challenges too*: legal and institutional restrictions, risk of disclosure, preservation of privacy/anonymity for information source, extra infrastructure for producing and consuming information, high costs for interoperability.
- *Cyber attack life cycle*: reconnaissance, weaponisation, delivery, exploit execution, installation of a remote access software, command and control, act on objectives. Defences and mitigations (preventive and detective – before the exploit execution; reactive and recovery – after) need to counteract at each of the attack stages.

- *Information sharing architectures*: centralised, peer-to-peer, hybrid. Type of the architecture should be determined by the information sharing requirements in the community. Factors to be considered: profiles of participants (capabilities, trustworthiness, etc.), level of commitment of participants and regulating agencies to support the community, type of information shared, information distribution profile (volume, speed, and frequency).
- *Preparation activities before launching an information exchange program*
  - *Goals definition*: stating the basic objectives that the organisation aims to achieve with this program, establishing a general scope of effort (what are the resources that the organisation can share, the resources that the organisation needs, potential partners and conditions under which the sharing will be performed), obtaining approval from management and legal teams and setting up a capabilities framework for information sharing specialists to operate within (organisational policies, etc.).
  - *Conducting an information inventory*: identification of information that supports key business functions and security operations. This information needs to be assigned an owner to manage it throughout the life cycle (also to determine its sensitivity and suitability for sharing). For this information the organisation needs to identify its physical and logical location, and the way it's stored.
  - *Establishing information sharing rules*: the organisation needs to set up the rules governing handling of sensitive information taking into account the risks of sharing it inside and outside organisation, and the legal and other requirements. Special challenge here is personally identifiable information (PII) and corresponding privacy challenges. Incident data need to be understood (by classification of its provenance and composition). For example, phishing email samples can be sensitive because they contain internal infrastructure information (internal IP address in the email header) but can also be private (contain PII). Table 5.1 summarises these considerations for various incident data types in a nice way. The rules for sharing also should identify which data types and content can be shared quickly with trusted partners; and how to track the information regarding the sensitivity of data to be shared.
  - *Considerations for joining the sharing community*: which organisations can be suitable partners (e.g., such that provide information complementary to the data collected internally, that provide actionable information, that provide information in a suitable format). Shared characteristics of organisations (business interest, industry sector, geographical boundaries, etc.) can be the basis for forming communities. Type of shared information, structure and dynamics within the community, and the cost of membership need to be considered also. Several trust models can be used within the community (specified also in NIST SP 800-39 on Managing Security Risk). Data sharing agreements, NDAs, SLAs, etc. might be necessary.

  - *Support for an information sharing capability*: there is a need for sustainable approach for information sharing that is actually incorporated into organisation's cyber security framework and with all necessary resources available.

- *Participation in sharing relationship*
  - *Engaging in on-going communication*: ways and costs of engagement need to be considered (online exchange vs. face-to-face meetings). Also within the organisation itself, coordinated incident management process and decision making for incident handling should be implemented on several levels, following NIST SP 800-39.
  - *Implementing access control policies for shared information*: for information produced within the organisation and externally obtained information the organisation needs to identify and enforce access control policies. These should be compliant with legal requirements and also the agreements within the community. The US-CERT Traffic Light Protocol is given as an example of sensitivity marking convention.
  - *Storing evidence*: evidence preservation should be done in accordance with best practices for data preservation following the chain of custody requirements and other laws applicable. More information on forensic techniques related to chain of custody and preserving information integrity are available in NIST SP 800-86 and NIST SP 800-61, revision 2.
  - *Consuming and responding to alerts and incident reports*: alerts provide technical information that can help to identify the level of exposure to vulnerability, potential impact, and mitigation steps. The organisation should have personnel and procedures to analyse alerts and to apply mitigations if necessary (the personnel should be able to evaluate the proposed mitigations and to decide to apply them or to look for other solutions).
  - *Consuming and analysing indicators*: the organisation should be able to interpret these indicators and to understand how to monitor for them in their own environment. The most efficient solution is to use in combination data from internal sensors with data on indicators received from external sources (if applicable).
  - *Records creating*: the organisation should produce the records throughout the incident response life cycle. The document does not really identify what is the best approach for sharing those.
  - *Performing local data collection*: data sources include log files, and alerts from all software (including Intrusion Detection Systems (IDSs) and antivirus products) and devices. The organisation should consider pruning the data (removing irrelevant information from the log files) and fine-tune its data collection system so that it does not collect too much of irrelevant information or impose a denial of service due to very aggressive collection. Partners from the sharing community can provide a good feedback whether the collected and shared data is useful and what can be further improved. There might be a need for

creating a structured knowledge base, especially for threat intelligence collection and representation. Wikis and structured databases can be useful here. The organisation needs also to design a data handling and retention strategy as a part of its data collection process, taking into account confidentiality and integrity requirements for this data.

- *Producing and publishing indicators*: indicators can be produced organically within the data collection process or through enrichment of indicators received from the sharing community. There are 3 types of indicators: atomic, computed and behavioural. Atomic indicators are data elements that cannot be further decomposed (e.g., IP addresses). Computed indicators are derived from other information (e.g., hashes). Behavioural indicators are composed of atomic and computed indicators; these can also include context information (e.g., patterns of network activity). organisations with basic network monitoring capabilities should be able to produce atomic indicators and some computed indicators from their data sources. More advanced computed indicators and behavioural indicators require better tools and analytics in place. It is important to collect and supply metadata with the indicators describing how it should be interpreted and used, how it was collected, what is the relationship with other indicators, sensitivity level, handling instructions, provenance, and what is the confidence level of the publisher. Incident data should be maintained in a version control system to identify and track the content and to monitor freshness. The sharing community should have a feedback mechanism and a way to remove erroneous or old data. Indicator format needs to be lightweight and easy to process.
- *Maintaining the sharing relationship*: there should be a constant discussion and feedback in the community on the incident response (was the collected threat intelligence and attack details useful in the incident response, were the proposed countermeasures effective, was information on this incident handling properly shared by the targeted organisation).

2.3.2. Relevance to the TRE$_S$PASS model sharing process

The document (Johnson et al., 2014) outlines some considerations that organisations need to take into account when participating in data sharing. Similar considerations apply to the model-sharing processes considered by TRE$_S$PASS. Benefits and drawbacks of participation in information sharing identified in the document are relevant also to model sharing. NIST has justified that sharing is beneficial in principle. One of the important points they make is that the value of data sharing is determined by the richness of the community and data available. If the community is not engaged in sharing or does not share much, then there will be no benefits to participate in it. Therefore, it is very important for TRE$_S$PASS to engage communities when establishing sharing infrastructures.

Considerations that organisations need to take into account are also relevant: confidentiality and privacy of its and others' information, proper data gathering and handling capabilities in place, engagement with trustworthy partners, etc. In this respect, TRE$_S$PASS

needs to consider specifically what are the requirements and procedures for model sharing, following the approach similar to the one described in this document.

The document identifies criteria for sharing community building: certain trust levels, relevance of the data (e.g., companies in the same industry sector). These are also very valid for model sharing.

In summary, TRE$_S$PASS will use lessons learned from this document as a starting point and will further identify details that are relevant to model sharing in the TRE$_S$PASS context: benefits, procedures, requirements, considerations, etc.

2.4. Sharing attack patterns

Attack patterns are a form of sharing security information based on abstractions of specific attacks. A master student supervised by Dina Hadžiosmanović and Wolter Pieters investigated the sharing of attack patterns. The text below is based on the thesis (Zhu, 2015).

Patterns represent events that are regular or repeated in some way. Commonly, people build patterns to encapsulate and reuse knowledge (Shirazi, Schaeffer-Filho, & Hutchison, 2014; Uzunov & Fernandez, 2014). In computer science, patterns are general, reusable solutions to commonly occurring problems (Bayley, 2014). Patterns have been found useful in diverse areas including software engineering, where this concept has received much attention both in academia and industry (Uzunov & Fernandez, 2014). The idea of patterns was originated in architecture from Christopher Alexander's architectural patterns for architecture design (Alexander, Ishikawa, & Silverstein, 1977). Then, it was transferred to software design as design patterns in the book Design Patterns: Elements of Reusable Object-Oriented Software (Gamma, Helm, Johnson, & Vlissides, 1995), and since there it entered the realm of cyber security as attack patterns and security patterns (Bayley, 2014).

Bayley (Bayley, 2014) analyses the commonalities between design pattern, attack patterns and security patterns and concludes that the instances must have the following features:

- Patterns are instantiatable; given a pattern, readers should be able to produce a list of actions for a concrete instance, the other way around, given an instance, readers should be able to easily identify which pattern it belongs to.
- A generalisation relationship exists between patterns that one is a subclass or superclass of the other.
- It should be possible to state the problem and prove that the pattern solves the problem.
- A pattern is a modification of an existing system, be it attacking with an attack pattern, defending with a security pattern.

Software developers need to accurately anticipate threats to be able to secure their software. Therefore they must think like attackers to identify security vulnerabilities (Barnum & Sethi, 2007). Attack patterns facilitate early mitigation of potential attacks that whenever a flaw is found, the software developer can either design for a safer system or generate security fortification strategies (Gegick & Williams, 2007). They provide a convenient way of encapsulating and reusing attack information (Shirazi et al., 2014). The concept of attack pattern is useful in teaching the software development community about exploiting software in reality and implementing proper ways to avoid attacks (Barnum & Sethi, 2007) The concept of attack pattern is based on the concept of design patterns (Blackwell, 2012; Gegick & Williams, 2007). "A design pattern captures the context and high-level detail of a general repeatable solution to a commonly occurring problem in software design". Therefore, it is a description of a solution that is reusable in many situations and it is not a low-level design that can be transferred directly into code (Barnum & Sethi, 2007).

In 2001, the term attack pattern was introduced in the paper Attack Modeling for Information Security and Survivability (Moore, Ellison, & Linger, 2001) and followed by several researchers in their work (Barnum & Sethi, 2007).

2.4.1. Attack pattern attributes

We now present how various works consider different attributes when describing attack patterns. All the attribute names are the original names from the information source. For instance, 'attack' from defenders' view and 'solution' (Fernandez, Pelaez, & Larrondo-Petrie, 2007) from the attacker's view refers to the same attribute of attack pattern. Among these sources, the CAPEC attack pattern schema list developed by Mitre Corporate layered the attributes into required, suggested and optional. We only focus on the required attributes; suggested and optional attributes (over 50) are not listed here. Moore et al. (Moore et al., 2001) use simple attributes to describe an attack pattern: name, goal, precondition, postcondition, attack. Barnum and Sethi (Barnum & Sethi, 2007) focus on attack motivation, pre- and post- attack conditions, this scheme uses the following attributes: pattern name and classification, attack motivation and consequence, solution and mitigation, attack description and method, attack prerequisites, required resources, attacker skill or knowledge, related vulnerability. Fernandez et al. (Fernandez et al., 2007) use the following attributes: name, problem, intent, consequences, countermeasure, context, known uses, related pattern. Blackwell (Blackwell, 2012) focus on the attacker and requires skills using the following attributes: name and classifier, perpetrator, motivation, intent, target, immediate impact, ultimate impact, execution, attack method, prerequisite, attacker resources, knowledge and skill. Mitre Corporation uses CAPEC summarising attack patterns with the following attributes: name, summary description, related pattern, pattern completeness, pattern abstraction, status. Table 2.1 compares different schemes according to the following aspects: attack motivation, precondition, method, postcondition, attacker, solution and references.

Table 2.1.: A comparison of attack pattern schemes

Attack Attribute	*Moore et al.*	*Barnum and Sethi*	*Fernandez et al.*	*Blackwell*	*Mitre Corp.*
Identifier	name	classifier	name	clasiffier	name
Objective	goal	motivation	intent	motivation	
Precondition	precondition	prerequisite	context	context	
Method	summary	description		execution, method	description
Postcondition	postcondition	consequence	consequence	immediate and ultimate impact	
Attacker		resources		skill	
Solution		mitigation	countermeasure and forensics		
References		references	known uses, relate patterns	references	related patterns, pattern abstraction, status

This table shows several common elements in attack patterns. In particular, there are properties of the attacker, such as objective/motivation and skill; properties of the system/context; and properties of the attack steps/methods themselves. These different categories are reflected in the sharing approaches in TRE$_S$PASS.

2.5. Conclusions

It is agreed that sharing security information is an essential part of cyber defence. *How* such information ought to be shared is less clear. Several proposals can be found, ranging from high-level initiatives and guidelines to low-level definitions of attack pattern attributes. The conceptual framework of the attack navigator provides some guidance as to how information sharing could be incorporated in TRE$_S$PASS. In the following chapters, we will discuss the incorporation of information sharing in TRE$_S$PASS in more detail. But first we will direct our attention to the modelling processes, in order to identify the opportunities for sharing in our innovative modelling processes more precisely.

3. Model creation

3.1. Strategy

The sharing of security information within TRE$_S$PASS cannot be seen separately from the model creation practices, as these form the context in which shared information can be reused.

From the beginning of the project, TRE$_S$PASS has emphasised process integration next to tool integration. We believe that the process of model development is as important as the modelling formalism and associated analysis techniques, and that stakeholders should benefit from research effort in model development. In this context, WP5 integrates the model development efforts in the project, ranging from use of existing development tools to elicitation of maps with LEGO.

In TRE$_S$PASS, models consist of system models and attacker models. Rather than working with implicit threat agents such as in traditional attack trees, TRE$_S$PASS enables flexibility by separating the threat model. Still, the main complexity resides in the map or system model, and that is what the model creation methods focus on.

In this chapter, we discuss the best practices for model creation in TRE$_S$PASS. The methods discussed have been selected from model creation efforts across the project, mostly in the context of large-scale and/or small-scale case studies. We have evaluated specifically why these methods count as best practices. Below, we discuss each method in a separate section, in a standardised format.

The final part of this chapter evaluates how these model creation practices have benefited from or could benefit from information sharing.

3.2. Actor-network analysis

3.2.1. Description of the method

In cybercrime, technological networks and criminal networks are mostly considered separately. TRE$_S$PASS invested in research (20%) on using actor-network theory to map hybrid cybercriminal networks, which explain both the technological and the human factors in the success of cybercrime (van der Wagen & Pieters, 2015). This can serve as one of the possible approaches for developing navigator maps.

3.2.2. How the method has been used

In a case study (20% foreground), van der Wagen and Pieters (2015) applied this framework to a botnet infrastructure, based on police files. This proof-of-concept study shows that Actor-Network Theory is a suitable framework for identifying actor roles and their contribution to security incidents / cybercrime in complex socio-technical systems.

Abstract: Botnets, networks of infected computers controlled by a commander, increasingly play a role in a broad range of cybercrimes. Although often studied from technological perspectives, a criminological perspective could elucidate the organisational structure of botnets and how to counteract them. Botnets, however, pose new challenges for the rather anthropocentric theoretical repertoire of criminology, as they are neither fully human nor completely machine driven. We use Actor-Network Theory (ANT) to provide a symmetrical perspective on human and non-human agency in hybrid cybercriminal networks and analyse a botnet case from this perspective. We conclude that an ANT lens is particularly suitable for shedding light on the hybrid and intertwined offending, victimisation and defending processes, leading to the new concept of 'cyborg crime'.

3.2.3. How the method could be used

In socio-technical systems, it is challenging to identify the different human and technical components and their interactions, which is needed to develop navigator maps. In this context, Actor-Network Theory and associated empirical analyses can be helpful, as they focus specifically on "opening the black box" of complex socio-technical systems, by following the actors and their actions. By identifying actors and their contributions to security, actor-network analysis can be an important first empirical step in navigator map development.

3.2.4. Why the method is a best practice

The symmetrical view of ANT forces researchers and stakeholders to consider human and non-human actors equally in the analysis of a security situation. This forms a first step in the empirical analysis leading to socio-technical navigator maps.

3.3. ArgueSecure

3.3.1. Description of the method

ArgueSecure is a brainstorming tool that allows stakeholders to build and maintain a list of high-level risks and respective countermeasures with the following structure:

Category: <A category of risks>

R1: <a risk>

(sword) C1: <Claim made by an attacker about the existence of an attack path>

A1.1: <An assumption of the claim>

A1.2: <Another assumption of the claim>

(shield) C2: <Claim made by a defender,that partly or completely defeats the attacker's claim by pointing out that an attacker's assumption is probably, or certainly, false>

A2.1: <An Assumption of the defender's claim, e.g. about a mitigation that alresdy exists or thst will be implemented.>

(sword) C3: <Renewed claim of the attacker that bypasses the defender's argument>

A A2.1: <An assumption of this renewed claim>

R2: etc.

Category: etc.

The process by which this list is populated assumes involvement of stakeholders of the Target of Assessment (or modelling target, ToA). By *identifying risks*, stakeholders inadvertently elicit *critical* components or assets they consider to be *at risk*.

3.3.2. How the method has been used

The tool has been used to conduct (distributed) high-level risk assessments of various infrastructures:

- IPTV (@ UT);
- Cloud (@ IBM);
- e-voting (@ CYB);
- ATM (@ TRE$_S$PASS meeting Zürich);
- a large bank.

3.3.3. How the method could be used

The list created can be used as input for any developing any sort of organisational model to be used in risk assessment, since these *critical* components and *valuable* assets should appear in any such model. Furthermore, the risks provided by stakeholders could be used to formulate attacker goals and subsequently construct attack scenarios to be analysed by the TRE$_S$PASS toolkit.

3.3.4. Why the method is a best practice

Stakeholders of the abovementioned infrastructures have indicated that the method is easy to use and provides useful insights into relevant actors and architecture components in risk. These can be used in following stages in the attack navigator tools and processes.

3.4. LEGO

3.4.1. Description of the method

The use of LEGO to support rich picture modelling within the TRE$_S$PASS project has enabled a much more comprehensive approach to gathering social, organisational and technical data (Peter Hall & Tanner, 2015; Heath, Coles-Kemp, Hall, et al., 2014). Physical models make the movement of data and practice very visible, creating an opportunity to reinforce the formal models that relate to these data movements. This approach is particularly applicable to Stage Zero risk assessments, enabling stakeholders to identify their goals and values while developing a comprehensive map of information sharing and protection practices.

3.4.2. How the method has been used

LEGO was deployed with our IPTV case study partners to co-construct a rich multi-perspectival and layered picture of data-sharing as a part of a proposed home payments service. Participants were asked to model (using the colours and language of ArchiMate) the central actors (yellow bricks), infrastructure (green bricks), data (blue bricks), and locations (pink tiles). The weighting and positioning of each element was collectively agreed upon. Patterns of data-flow were observed across spaces of trust, and the model was enlivened by the participants' use of LEGO avatars to represent the central actors and the control strengths of selected points along data-paths. In particular, the second session was treated as an opportunity for the group to reflect upon and remodel the weaker parts of the service design, which were enhanced with further bricks to that effect.

Subsequently, hand-made line drawings of the LEGO models were made, tracing out the paths of data-sharing across the envisaged service as it had been constructed physically. To a degree, the drawings abstract away from the detail of the model, by outlining the distinct fields of practice and resilience that can be observed around the compounded nodes of infrastructure, data, and actors. This redrawing of the model was carried out in order to facilitate the analysis of the naturally occurring protection points of the data-flows within the service, those that are afforded by the surrounding social practices, for example.

3.4.3. How the method could be used

The combined LEGO and drawing method responds to the interdependencies of the scenarios under discussion. This approach has the potential to be translated into digital tools for analysing socio-technical practices, the analyst using a visual component to order the data. Drawings and diagrams are a powerful tool at the disposal of interface designers, researchers, and users, that can capture the complex and hard-to-reach patterns of social practice. Moreover, the methods visualise and find new ways to encode the persistence and change of practices. Importantly, for risk management tool design, they help to make visible and give pattern to the factors that impact upon the willingness of actors to share and protect their data.

3.4.4. Why the method is a best practice

The strength of this approach to using LEGO in modelling lies in its support of rapid take-up by mixed groups, and the way in which it can be used to build a consensus around what constitutes a desirable representation of the infrastructure under discussion with relative speed compared to traditional business modelling tools. In addition, it helps to avoid the consequential flattening of the environment that accompanies such tools, enabling a very rich topology to be created from simple elements. The participative nature of the process supports a sense of ownership of the outcomes amongst the participants.

3.5. Other physical modelling

3.5.1. Description of the method

Creating an accurate TRE$_S$PASS model requires in-depth knowledge of ToA and its environment, as well as of the modelling language. Knowledge of the language cannot be assumed to be present in domain experts.

To facilitate the creation of a TRE$_S$PASS model, we propose to communicate with domain experts using a tangible representation of TRE$_S$PASS concepts. This stimulates participation of and collaboration among domain experts and modelling experts, and evidence suggests (Ionita, Wieringa, Bullee, & Vasenev, 2015) that the resulting models are more accurate.

We defined a tangible representation of the TRE$_S$PASS concepts described in D1.3.1 (The TRE$_S$PASS Project, D1.3.1, 2013). Table 3.1 shows the resulting mapping.

Figure 3.1 shows a TRE$_S$PASS tangible modelling toolkit and Figure 3.2 shows part of a tangible model created using these conventions.

Table 3.1.: Mapping of concepts to representations

Concept	Software representation	Tangible representation
Actor	Stickman	LEGO®character
Asset (item)	Solid circle	LEGO®item[a]
Asset (data)	Dotted Circle	LEGO®mini-brick[b]
Location (network)	Box (green)	Hexagonal card
Location (physical)	Box (yellow)	Box
Location (device)	Box (Blue)	Card
Access policy	Text-box	sticky-note
Relationship (position)	Solid line	Physical overlap
Relationship (possession)	Dotted line	Physical attachment
Relationship (containment)	Directional arrow	Physical overlap
Relationship (connection)	Bi-directional arrow	Line

[a] A LEGO®item resembles a real-world object and can be placed in a LEGO®characters' hand

[b] A LEGO®mini-brick is the smallest LEGO®brick available, usually of circular or cylindrical shape and can be placed on a LEGO®characters' head

Figure 3.1.: Tangible TRE$_S$PASS modelling kit

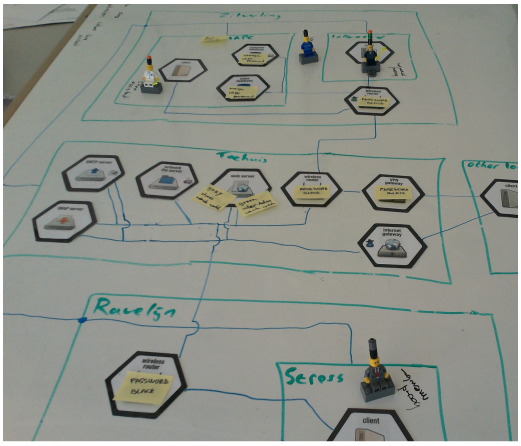

Figure 3.2.: Part of a tangible TRE$_S$PASS model

3.5.2. How the method has been used

The method has been used in a student experiment aimed at measuring its usability when compared to the existing software tool (Architect) as well as in a focus group with practitioners from BiZZDesign. Initial results are promising, but scalability of the approach seems to be restricted to small-medium sized models due to physical limitations.

3.5.3. How the method could be used

To enhance the utility of a tangible modelling approach in TRESPASS use-cases, we plan to explore tool support for automatic conversion from a tangible model to an abstract representation of the model and vice versa. For example, via an over-head camera.

3.5.4. Why the method is a best practice

Conceptual models, such as the TRESPASS model represent social and technical aspects of the world. To build these models, knowledge might have to be collected from domain experts who are rarely modelling experts and don't usually have the time or desire to learn a modelling language. Furthermore, the TRESPASS language itself is complex and cumbersome, making usage and adoption a challenge. Using physical tokens to represent the model can help overcome this challenge.

3.6. ArchiMate and Architect

3.6.1. Description of the method

The ArchiMate standard, maintained by The Open Group (The Open Group, 2013), describes an open and independent modelling language for enterprise architecture that is supported by different tool vendors and consulting firms. The ArchiMate *core language* provides concepts to model the high structure and behaviour of the business, applications and (information) technology of an organisation, as well as the relationships between these aspects. The main focus of the ArchiMate language is on the *coherence* of the different domains. For the details of each domain, ArchiMate elements may be linked to models in other more specialised languages (e.g., BPMN models for the details of business processes or UML models for the details of IT applications and infrastructure, but also security-related models such as the TRESPASS model).

Since version 2 of the standard, two extensions to the language have been included, one for modelling the business motivation for the architecture, and one for modelling the implementation and migration aspects. The motivation extension is used to express the stakeholders and their goals, as well as the principles and requirements that should be fulfilled by the design of the organisation represented by the architecture model.

Figure 3.3 summarises the main elements and relationships of the ArchiMate language, both core and extensions, positioned in the ArchiMate framework. This overview does not show the complete set of elements, but the most commonly used elements have been included.

A new update of the ArchiMate standard is under development, expected to be published in the first half of 2016.

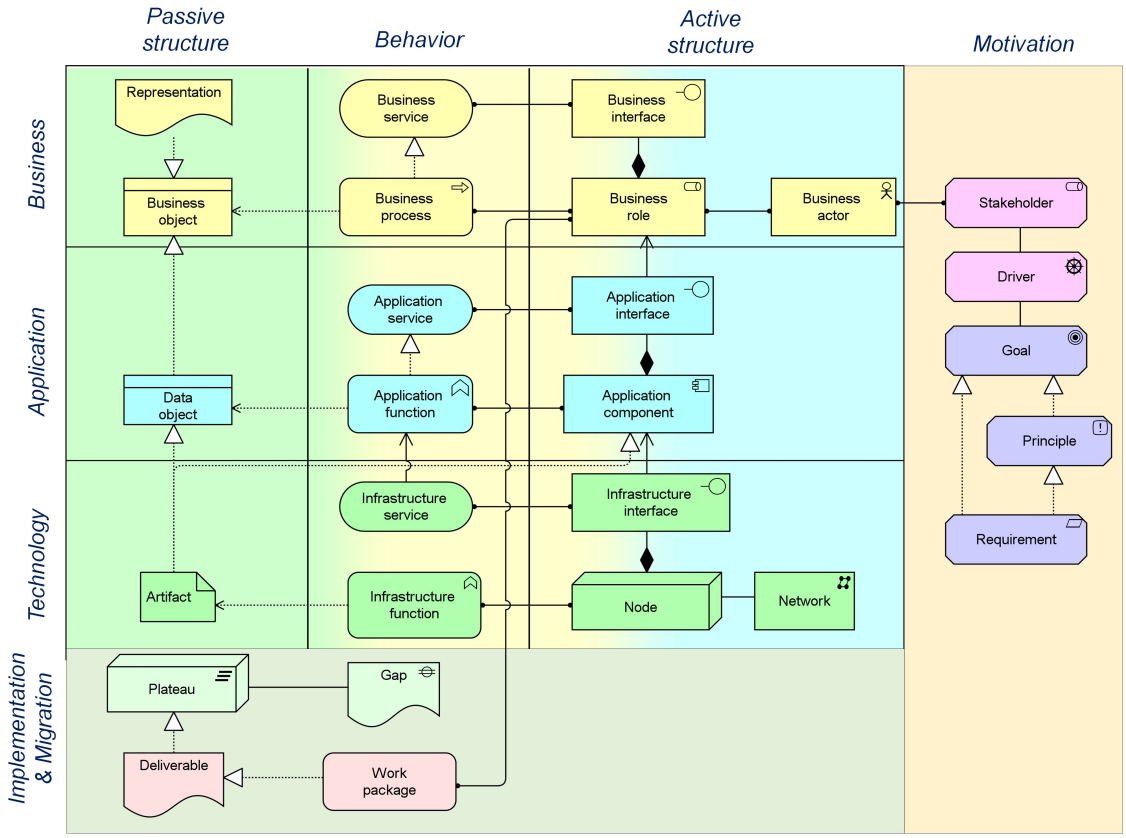

Figure 3.3.: Overview of the ArchiMate language

A common best practice when using the ArchiMate language is the use of *viewpoints* and *views*, in conformance with the ISO/IEC/IEEE 42010 standard (ISO/IEC/IEEE, 2011). Given a single underlying integrated model, a viewpoint specifies a subset of that model, with an associated (graphical) representation (either using the standard ArchiMate symbols or some other kind of visualisation, depending on the purpose of the viewpoint). A viewpoint is aimed at a stakeholder or set of stakeholders that has an interest in the system that is modelled. Figure 3.4 sketches some simple examples of the use of viewpoints. The model repository contains a complete, coherent model of the enterprise architecture, with business processes and actors in the business layer, application components and services in the application layer, and nodes, devices, system software and artifacts in the technology layer. An application architect may work on a view that only shows the business layer, while an infrastructure architect may work on a view that only shows the technology layer; a process owner may be presented with a view that shows his process, together with the applications and technical infrastructure that support this process. Other ways to present information from the model include a cross-table that shows which of the applications support which of the business processes, or more “clipart-like” pictures to represent the model elements.

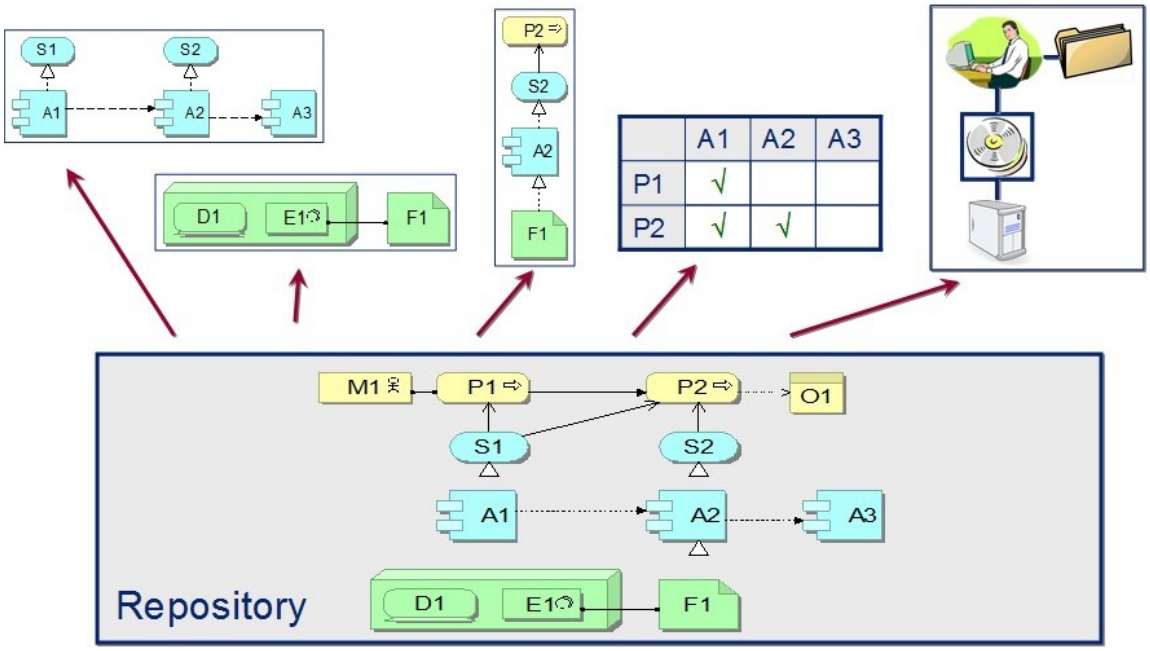

Figure 3.4.: Examples of the use of viewpoints in ArchiMate

3.6.2. How the method has been used

The ArchiMate language has been used extensively for modelling enterprise architecture in general. Specifically for risk and security, a white paper has been published by The Open Group (Band et al., 2015), describing how these aspects can be expressed with the ArchiMate language. This paper proposes specialisations of ArchiMate model elements, both from the core and the motivation extension, to model the outcome of a risk assessment, and the requirements for security controls. This so-called 'risk and security overlay' of ArchiMate has also been implemented in an innovation release of BiZZdesign Architect. Figure 3.5 summarises the element of this overlay, with the symbols as they are used in Architect.

Some experiments have been carried out to import the results of a network penetration test (pentest) in BiZZdesign Architect. Based on the output of a vulnerability scanner, a part of the technology architecture is generated, possibly integrated in a pre-existing enterprise architecture model. Also, an overview of the identified vulnerabilities is generated (using the above-mentioned ArchiMate risk and security overlay), linked to the core elements in the architecture. The pentest results, as well as the business impact of the vulnerabilities, can be visualised in a number of ways (e.g., as 'colour views' or as heatmaps).

3.6.3. How the method could be used

Many medium to large information-intensive organisations have ArchiMate models of their enterprise architecture in place, which often includes a description of their (technical) in-

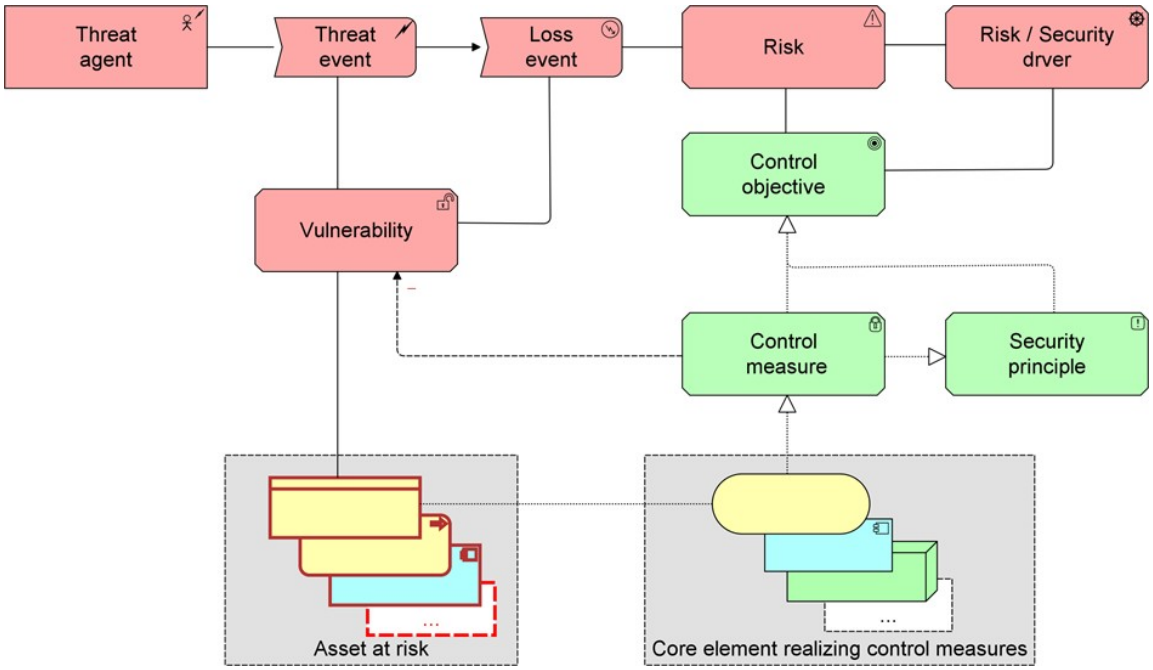

Figure 3.5.: Model elements of the ArchiMate risk overlay

frastructure. In those cases, relevant data can easily be collected. Within TRE$_S$PASS, we have demonstrated that the infrastructure as defined with the ArchiMate core language can be mapped to elements of the TRE$_S$PASS model. Figure 3.6 shows a simple example. Therefore, ArchiMate can be used as a 'front end' for the creation of a TRE$_S$PASS socio-technical security model, which is useful for users already familiar with ArchiMate, and possibly with the enterprise architecture of their organisation already modelled in ArchiMate. Conversely, if the reverse mapping is also available, analysis results can be fed back in the ArchiMate model and visualised as appropriate.

Another option is that risk assessment results that have been documented in an ArchiMate model (motivation extension) may be used as input for attacker models. To find out to what extent this is possible, further research is required.

3.6.4. Why the method is a best practice

The ArchiMate standard is an open, vendor-independent standard for enterprise architecture modelling, which has been adopted globally as the language of choice for many enterprise architects. Several enterprise architecture tools, both commercial and freeware, support the language. Within the project, the tool Architect of project partner BiZZdesign is used. Although this is a commercial tool, a hosted version is freely available for project members. Architect natively supports the ArchiMate language, but is also configurable for other languages (an editor for the TRE$_S$PASS model in Architect has been created within the project).

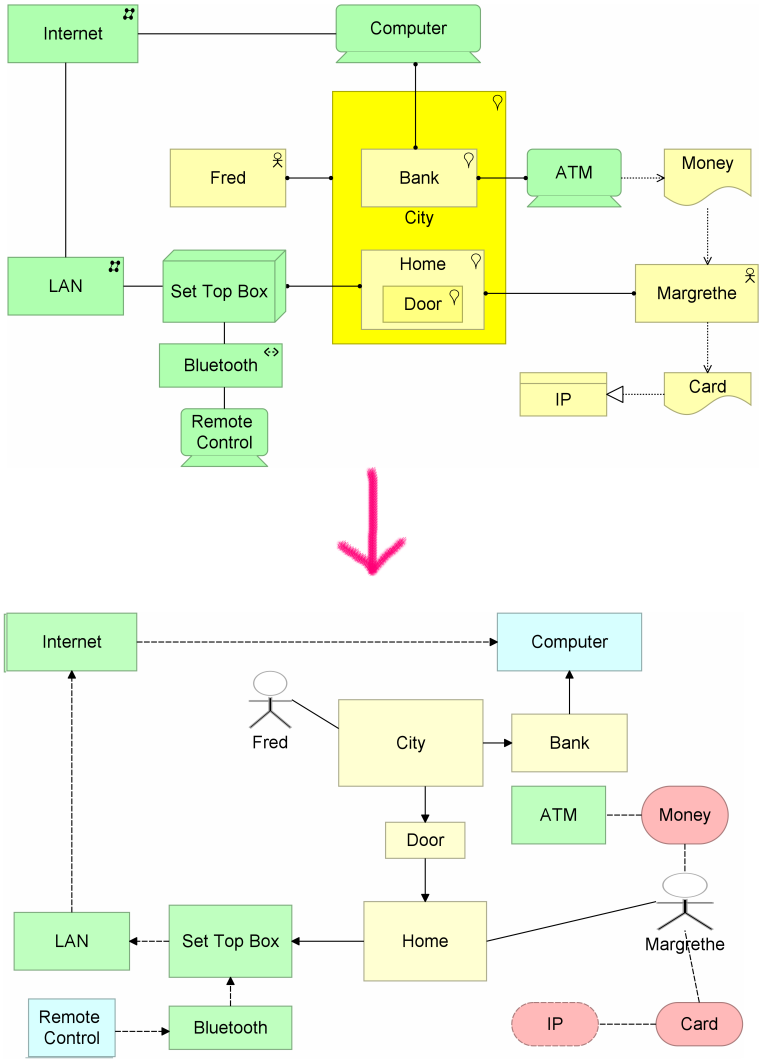

Figure 3.6.: Example ArchiMate to TRE$_S$PASS model mapping

The ArchiMate language is particularly suited to create an integrated model of the organisation and its environment, including both business and technology aspects. Therefore, the language is also very suitable for an integrated approach to risk and security management. Because of the precise definition of the language elements and relationships, ArchiMate can be used as a basis for different kinds of analysis and visualisation. The viewpoint mechanism as described by the standard helps to structure the models and create stakeholder-specific visualisations. BiZZdesign Architect has several built-in analysis and visualisation options, as well as a powerful scripting language is available to implement other types of analysis, mappings, etc.

3.7. The TRESPASS user interface

3.7.1. Description of the method

The Attack Navigator (AN), as described in detail in (The TRESPASS Project, D6.3.1, 2015), is the online TRESPASS environment where all tools developed within the project can be viewed, accessed, used and connected. A main part of the user interface is one of the project's major tools, the Attack Navigator Map (ANM), a tool that predicts and prioritises attack scenarios based on a model of the system or organisation concerned. It can also be used to judge the effect of countermeasures, by re-running the analysis with an adapted model. The model takes the form of a navigator map and a set of attacker profiles. The navigator map represents the system cartographically, displaying connections between the elements as potential steps that an attacker could take. These steps are annotated with relevant variables such as difficulty and cost. The attacker profile collects relevant characteristics of an attacker, such as skill, resources, motivations / goals, and initial access. For a combination of a map and a profile, the system will calculate routes for the attacker across the map that provides utility to the attacker, and visualises where vulnerabilities are in the mapped environment. The ANM does this by making use of many of the tools developed within the TRESPASS project. Each tool can be used separately, but also in a chain, where the export of one tool functions as the input for another tool.

The TRESPASS workflow goes through four stages and the interface of the ANM takes this into account. The stages may have various activities, not all of which are required for every risk assessment scenario. The main stages are: 1) data collection stage, 2) model creation stage, 3) analysis stage, and 4) visualisation stage.

Attack Navigator Map The Attack Navigator Map is the central tool in the model creation stage of the TREsPASS tool chain. It is where data from different sources (data collection stage) come together, to be incorporated in the model. The model itself is graphically represented as a 'map', which visualises the threat landscape of the system, properties of its components, and the relations between them.

Data from the data collection stage is either imported or added manually. Most technical data (such as data from automated network discovery processes, for instance) is converted if needed, and can then be loaded via an import function. Social data - knowledge, in many cases - requires manual editing of the model.

The basic building blocks for constructing a model come from libraries of single components, or of prefabricated model fragments (groups of components with relations), such as the model pattern library. These libraries will contain commonly used patterns, that can be used as templates to rapidly build the basic structure, which can then be refined and tweaked.

The underlying data structure is a directed graph of nodes (components with properties) and edges (relations between those components). Future work on the ANM will concentrate on refining the visual language of the map, to make it look less like a network graph

and more like an actual landscape. We envision a more abstract representation with a look and feel that is more specific to components domains and properties. Furthermore, it will be based on the principle of showing details only selectively, when they are relevant to the user.

Models do not have to be created in the ANM exclusively. Diversity in the choice of tools is desirable. Therefore, if a user prefers a different modelling tool or does not want to change an existing workflow, that should not be a problem, as long is the model is or can be converted to the TRESPASS XML format before importing it into the TRESPASS tool chain.

Once done editing, a scenario (the model + attacker goals) is constructed, and passed on - along with the attacker profile(s) - to the next step: the attack generation stage.

Once the analysis finishes, its results are visualised, with the possibility for the user to rank and filter the attacks. In a split screen dashboard, there will be multiple other views besides the list of attacks. Overview will include showing the attack traces in the context of the intermediate attack tree, or highlight the relevant parts of the model/ANM that play a role in the attack(s).

3.7.2. How the method has been used

The user interface is still under development, and will be extensively tested in M36-M48 of the project.

3.7.3. How the method could be used

The ANM plays an important role in all the four phases of the TRESPASS workflow

1. Data collection stage: Gathering relevant data in the security domain is often quite difficult for several reasons: company knowledge, privacy, legal issues, etc.. The ANM can also function as a data gathering tool, for refined information on locations, assets and actors, and all of their properties. Through the ANM all properties can be adapted and changed in a user-friendly manner.
2. Model creation stage: The ANM can be used to hand-craft a map of an organisation, with locations, assets and actors, in the physical and virtual domain. A user can also use the library elements that makes it easier to build more complicated maps. Library elements contain typically multiple elements, for instance a complete room or even floor of a building, with doors, windows, etc.. A library element can also contain virtual infrastructure, as a complete cloud centre with multiple VMs en firewalls. All properties of each element have preset values, but can be changed according to a users need. Next to this, the ANM allows also for importing existing graphs of an organisation, for instance made in Architect, exported to the TRESPASS XML format and be used as a basis for applying and creating the TRESPASS model

3. Analysis stage: The model that comes out of the model creation stage can be analysed with a number of tools that are developed within the TRESPASS project. Each tool has a specific way of looking at the outcomes and highlights specific aspects. The ANM can present all these analyses so that the user has tools to make informed decisions.
4. Visualisation stage: To help making decisions easier and understandable, the analysis results are visualised in a clear and appealing manner.

3.7.4. Why the method is a best practice

Building and constructing navigator maps for medium or large companies is not an easy task and takes a long time, which is why the AN and the ANM makes use of the latest insights in user friendly interfaces and takes the user experience as a centre point that should make it easier to really use the interface. The wizard to construct the maps plays an important role in this, since a user can, with minimal training, start immediately and is guided step-by-step through all the elements she needs to apply. All logical interactions are always one click away, and easily editable.

The flexibility to import XML-schemas from other programs as Architect, or via a graphML program, is aimed at connecting to existing workflows and not enforce new ways of working. The fact that the AN is consisting of a set of loosely coupled tools means that each tool can be maintained and updated separately, without having to update the entire system. This makes deployment faster and less complex. The ANM truly functions as an interface between all the different models, tools, analyses and visualisations developed in the TRESPASS project, providing powerful features in a relatively simple package.

3.8. Model development case: e-voting

3.8.1. Description of the method

A case study was executed on applying the navigation metaphor to the Estonian electronic voting system, including development of a map based on available data, such as black market prices of vulnerabilities (Verbij, 2014).

Whereas most of the scientific literature is focused on highly theoretical environments in which e-voting schemes operate, the majority of the more practical research does not provide for quantified and practical results in a realistic setting. This research, however, fills this gap by establishing a quantified framework for reviewing and objectively comparing e-voting schemes in practice. The proposed framework in this research first establishes an exhaustive list of all possible attacks on the e-voting scheme, beyond initial traditional research into the protocol, the cryptography and the implementation. Subsequently, the framework addresses each of these identified attacks in terms of effort for the attacker:

how much time and money does an attacker need to pull off the attack? Thereafter the attacks are categorised according to the Dutch requirements for voting schemes, after which they can be compared to either a baseline proposed by politics or to other schemes. The final step of the framework helps in mitigating the attack vectors and assessing the impact of differences in implementation details of these schemes. As the framework is circular, all steps can be repeated to allow for a thorough analysis and mitigation of potential risks.

3.8.2. How the method has been used

We also presented a limited case analysis on the Estonian e-voting scheme in order to show how the framework can be used in practice. Results already show astonishing attacks, by means of which it would only cost $40,000 to get one seat in the Estonian parliament. Both the framework and the case analysis have been validated by three professionals in the discipline of IT Security; e-voting research; and the Dutch voting practice.

3.8.3. How the method could be used

While a few improvement points were identified, the framework is considered to be a strong method for realistic risk identification in e-voting schemes. Quantifying these risks in terms of effort for an attacker allows for effective risk management and strong mitigation strategies based on cost-benefit considerations. Especially the adaptive attacker model, the modular approach and the ability to test the effectiveness of implementation details are very well received.

3.8.4. Why the method is a best practice

As a conclusion, this framework provides for an effective and structured additional research method when deciding to adopt e-voting for elections. Furthermore, this research fuels the current debate about e-voting.

3.9. Implications for sharing

These methods highlight various sharing needs in the process of developing and using TRESPASS models.

Firstly, there is a question of how to define properties of agents / actors in the model. This holds in particular for the adversarial agents, or attackers. Although attacker profiles are fairly simple, many users may lack the necessary knowledge to develop them from scratch. Therefore, standard attacker archetypes could be shared in a library of attacker profiles (Chapter 5).

Table 3.2.: An overview of model development methods

Method	Input	Output	Use
Actor-network analysis	Complex system	Actors and relations	Identifying model entities
ArgueSecure	Stakeholder discussion	List of risks as structured arguments	Identifying model entities
LEGO	Stakeholder workshop	LEGO physical model	Initial model development (satellite view)
Other physical modelling	Stakeholder workshop	Other physical model	Initial model development (satellite view)
ArchiMate and Architect	Expert input, possibly initial (non-formal) model	Formal (ArchiMate) model	Model formalisation
The TRE$_S$PASS user interface	Expert input, possibly initial (non-formal) model	Formal (TRE$_S$PASS) model	Model formalisation

For model creation activities, it would be helpful if users can start from a simple model template. In addition, reusable parts of models would speed up the model creation process. To this end, a library of model patterns is needed. As model creation happens in very different settings, the sharing of standard model components may depend on the method. For example, sharing standard components in LEGO will take a different form than in Architect. Still, for the digital creation tools, the XML format enables reuse across different model creation tools, as long as these tools can import as well as export the TRE$_S$PASS model format. We will discuss the model pattern library in Chapter 4.

As it is unrealistic to specify the models to a level of detail that can produce the smallest possible attack steps, it is beneficial to have a library of attack patterns, that can refine high-level attack steps into more detailed possibility for the attacker. For example, a possible social engineering attack could be refined into different available social engineering tactics. The attack pattern library will be discussed in Chapter 6.

3.10. Conclusions

This chapter described the best practices for model creation that have been developed in TRE$_S$PASS (Table 3.2). In particular, such methods need to achieve two goals: extraction of the relevant knowledge from the stakeholders (joint work with WP2), and formalisation of such knowledge in formal models (joint work with WP1). For the latter goal, it is important to enable reuse of components, such that the formalisation does not have to be developed from scratch. To this end, TRE$_S$PASS has developed libraries for information sharing.

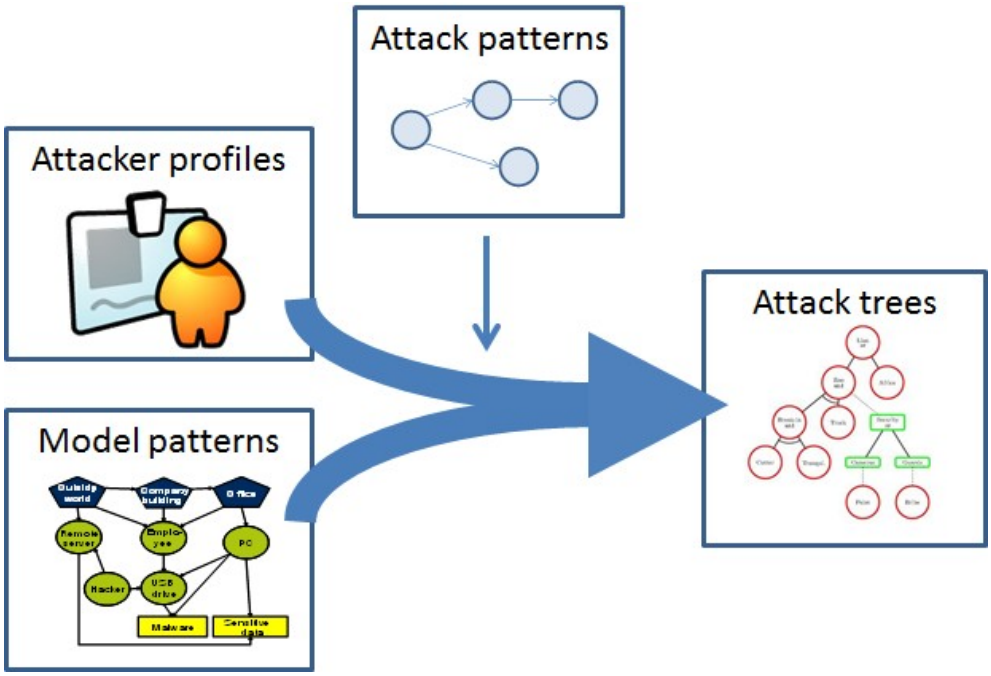

Figure 3.7.: An overview of TRESPASS sharing modules

In TRESPASS, there are three key parts of the model and its analysis: the attacker, the system, and the attack steps / actions. These key parts have formed the basis for our data sharing approach (Figure 3.7). For representing the attacker side, attacker profiles are available. For representing the system side, model patterns are available. When deriving possible attack actions from attacker plus system description, an attack pattern library is available for reusing possible action sequences. Finally, entire attack trees can be shared. These types of sharing will be discussed in the following chapters.

4. Model Pattern Library

4.1. Strategy

The Model Pattern Library (MPL) was drafted in The TRESPASS Project, D5.3.1 (2013), but its instantiation was left unclear due to the early nature of the document. It is, however, an integral part of the model creation described in Chapter 3, since it enables modellers to create models more quickly using prefabricated, shared elements, and it also supports the attack pattern library, since it ties model elements to shared properties of these elements.

The Model Pattern Library was initially not considered in the TRESPASS project, but has been identified as an important element in the requirements identification process (The TRESPASS Project, D6.2.2, 2015) as an outcome of discussions on sharing of models and model components within the project. The MPL has been designed as a library for sharing parts of TRESPASS models, that may be anything from individual components to complete (sub-)models. All of these can be represented in some standard format, in our case XML.

First, we will discuss the possibilities for reusing existing knowledge databases and modelling languages as a basis for MPL elements. Thereafter, we will outline in more detail which types of elements can be stored in the MPL, and what the current architecture is.

4.2. Reusing knowledge

One of the main principles of the model language development is its simplicity. The core language is kept rather small, and extensions (The TRESPASS Project, D1.3.2, 2015) will be developed to handle more complex, domain-specific situations. The extensions will need to be stored in some library and MPL is exactly a suitable instantiation for it.

Part of the MPL is a library of model components, where models of some frequent real-world artefacts may be stored, for example, ID cards, doors, and servers. These components come with the respective model components, that become part of a model, and annotations and corresponding attack trees stored in the Attack Pattern Library as described in Chapter 6.

As discussed above, and as is the case for all shared components in TRESPASS, these components can be created by a modeller for its own organisation, or they can be shared

between parties. The former case represents specific knowledge about a certain component that is modelled, the latter represent, for example, model components that a manufacturer could share to describe its security doors and their properties.

A third option that is especially relevant for model components is to translate model components from existing approaches such as CORAS into the TRESPASS model language, in order to reuse existing modelling efforts. This reuse could occur through a direct import or adapters.

In Appendix B we list existing models and modelling languages that could be used to import existing model components into the TRESPASS project. In addition, The TRESPASS Project, D1.1.2 (2015) provides a more thorough overview of existing modelling languages.

4.3. The library

Like for other modelling languages, TRESPASS models can be shared. This holds both for entire models and for smaller parts of those models. These smaller parts can be elements of real models that are thought to be reusable, or templates with limited details that can be tailored depending on the available information in a specific modelling context. In addition, real-world knowledge may lead to specific modelling elements that can be reused, such as routers or smartcards. This leads to a variety of sharable elements, based on their level of abstraction as well as their size. For example, one could share:

- A model of a building that was used for security analysis;
- A part of this model representing a floor in this building (corridor, offices, assets, employees);
- A template for a floor in a building without annotations or with default annotations; or
- Modelling elements such as offices.

Clearly, modelling elements are the least sensitive to share, whereas entire models will typically not be shared based on confidentiality considerations. As partial models or templates inherently use the same XML format as entire models, the sharing of such elements is easy. General model patterns can be added to an existing model. Connections to the entities in both parts of the model can then be added manually. This is supported by the TRESPASS user interface developed in The TRESPASS Project, D6.3.1 (2015), which contains the current library of model patterns. From the interface, users can load full models, start with templates, or add standard modelling elements. The current Model Pattern Library contains:

- A sample model for a cloud company;
- Templates as a quick start for models;
- A set of standard modelling elements such as rooms, computers, etc.

4.4. Conclusions

In TRESPASS, models or components thereof can be shared via the Model Pattern Library. Together with attacker profiles (next chapter), these provide reusable model components that can be used to create navigator maps, from which attack scenarios can be generated. The user interface enables reuse of models, templates and standard components. Based on the use of the tools in case studies and elsewhere, the MPL will be filled further in the last project year.

5. Attacker Profile Library

5.1. Strategy

One of the key departure points of the TRESPASS processes is that properties of attack steps are dependent on properties of the system (map) as well as properties of the attacker. To this end, selecting relevant attacker profiles forms a key step in the processes. In this chapter, we outline the relevant work in the area of attacker profiling for threat assessment, and we discuss how we implement these ideas in the TRESPASS attacker profile library (APriL).

5.2. Agent-centric threat assessment

Several previous studies have investigated the use of attacker profiles in security risk management. A master student supervised by Dina Hadžiosmanović and Wolter Pieters investigated the use of threat landscapes in risk assessment, in particular focusing on threat agent libraries. The following text is based on his thesis (Pushpakumar, 2015)

There are different threat assessment methodologies which assess threats by focusing on the analysis of attackers. For example, Sandia National Laboratories, the Department of Homeland Security (DHS), the Federal Network Security (FNS) developed the Operational Threat Assessment (OTA) methodology which estimates the current threats faced by a system (Mateski et al., 2012). The OTA is designed to provide an efficient threat estimate which is consistent with respect to different organisations and analysts. The method focuses on characterising cyber threats using the characteristics of different agents. In particular, the methodology uses a General Threat Matrix (GTM) to characterise and differentiate threats against targets of interest across their commitment and available resources. The methodology however, does not try to identify the threat agents, and concentrates on building a GTM , using information about the attributes of agents - commitment (includes intensity, stealth, time), and resources (technical personnel, knowledge, and access). As an advantage, the methodology reduces the effect of personal bias and preconceived notions of assessors. However, as a disadvantage, the threat metrics can be difficult to identify, delimit and quantify. The time taken to perform the assessment depends on the boundary of the system under consideration.

A similar methodology, Intel – Threat Agent Risk Assessment (TARA) was discussed in Casey et al. (Casey, Koeberl, & Vishik, 2010) and Rosenquist (Rosenquist, 2009). The

predictive output from TARA helps in informed decision making at every level of management, which even non-expert audiences can understand. In contrast to OTA, this methodology directly identifies different types of attackers, their methods and objectives. In particular, TARA uses the concept of object and method libraries which closely correspond to TRE$_S$PASS libraries. In the next section we explain TARA in more detail.

5.2.1. TARA

TARA (Threat Agent Risk Assessment) methodology focuses on analysing threat agents which pose the greatest risk. TARA is developed by Intel to identify threat agents that are pursuing objectives which are reasonably attainable and could cause unsatisfactory losses (Rosenquist, 2009). The methodology analyses agent motivation and the likely methods they will employ. The methods can be cross referenced with existing vulnerabilities and controls to determine the most likely threat scenarios. The methodology uses three main libraries, Threat Agent Library (TAL), Common Exposure Library (CEL), and Methods and Objectives Library (MOL).

The TAL library captures different types and characteristics of threat agents. Namely, the TAL library currently consists of 22 agent archetypes which are characterised by eight attributes:

- Intent – Defines whether the agent intends to cause harm. Based on the intent, the agent can be hostile or non-hostile.
- Access – Defines the extent of agent's access to the company's assets. Based on access, the agent can be internal or external.
- Outcome – Defines the agent's primary goal.
- Limits – Defines the legal and ethical limits that may constrain the agent.
- Resource – Defines the organisational level at which the agent typically works.
- Skill Level – Defines the special training or expertise an agent typically possesses.
- Objective – Defines the action that the agent intends to take in order to achieve a desired outcome.
- Visibility – Defines the extent to which the agent intents to conceal or reveal his or her identity.

For example, the objective of *damaging* an asset is related to an *disgruntled employee* rather than to agents such as *thief, spy* etc. Similarly, while *anarchist* and *thief* need minimal resources to accomplish their goal, *competitors, spy's* and *internal employees* are characterised by high operational resources. TAL enables risk managers to uniquely define agents, and also select agents by first identifying the attributes that an agent must possess in order to represent a threat. For e.g. if the certain damages to assets can only be caused by an agent with internal access, we can focus on the agents with internal access for that particular asset. Instead of selecting agents based on names, the risk

managers can first identify the attributes required to affect particular assets and identify relevant threat agents possessing similar attributes (Casey, 2007). An important advantage of TAL is that it leaves room to accommodate the evolving threat agents due to changes in economic, political, societal, and technological trends.

The CEL library enumerates known information security vulnerabilities and exposures at Intel and is not publicly available. In practice, this library can be replaced by any known database of known vulnerabilities. The MOL lists known threat agent objectives – what they want to accomplish and the most likely methods they employ to reach their objectives. For example, a competitor as threat agent will likely copy and expose internal information about the company, and not focus on damage or takeover. On the other hand, a disgruntled employee will focus on making a personal gain by damaging, altering or denying access to critical information.

By combining the methods (from MOL) and types of agents (from TAL), an assessor captures potential threat agents and their operational modes.

5.3. TRESPASS attacker profiles

We believe that TARA is the most extensive threat agent framework, and we therefore use it as the basis for attacker profiles. In TRESPASS, we focus only on malicious threats, so we assume threat agents to be hostile in *intent*. The *access* of agents is part of the navigator map: the attacker is assumed to start at a certain place on the map, and will have certain credentials. The *outcome* and *objective* are considered part of motivation, and our scientific work on including these is discussed in section 5.4. *Visibility* could be an extension in future work, related to the response of threat agents to the likelihood of detection and associated punishment. We currently do not attempt to include *limits*, as this is outside our core area of expertise.

In this section, we focus on the remaining variables *resources* and *skill*. We distinguish between monetary resources (simply called resources) and the time that the attacker has available. Our current implementation of attacker profiles is based on Lenin, Willemson, and Sari (2014); Pieters, Hadžiosmanović, Lenin, Montoya, and Willemson (2014) and is described in The TRESPASS Project, D3.3.1 (2013). Below, we discuss how the attacker profiles are used in attack tree analysis.

Let us have a set of all possible elementary attacks $\mathcal{X} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n\}$, and a Boolean function $\mathcal{F}$ corresponding to the attack tree. *Attack suite* $\sigma \subseteq \mathcal{X}$ is a set of elementary attacks which have been chosen by the attacker to be launched and used to try to achieve the attacker goal.

In our research (see Lenin et al. (2014)) we use the following attacker properties:

1. *Budget* $t_b \in \mathbb{R}$ – the monetary resource of the attacker, measured in currency units.
2. *Skill* $t_s \in \{L, M, H\}$ – the skill level of the attacker, measured on an ordinal scale (Low/Medium/High).

3. *Time* $t_a \in \{S, MT, HR, D\}$ – the available time resource of the attacker, measured on an ordinal scale (Seconds/Minutes/Hours/Days).

The attacker properties outlined above define function $\mathcal{P}_f(\sigma)$, which returns *true* iff:

1. $t_b \geqslant \sum_{i=1}^{n} \text{Cost}(\mathcal{X}_i)$,
2. $\forall \mathcal{X}_i \in \sigma : \ t_s \geqslant \text{Difficulty}(\mathcal{X}_i)$, and
3. $\forall \mathcal{X}_i \in \sigma : \ t_a \geqslant \text{Time}(\mathcal{X}_i)$,

where Cost, Difficulty and Time refer to the corresponding estimates of the elementary attack $\mathcal{X}_i$. Essentially, attacker profile is used to prune the potential space of attack suites, since the profile describes the limitations put onto the attacker.

Possible extensions of this set of parameters was suggested by the attacker interviews reported in (The TRESPASS Project, D5.1.2, 2015). It was suggested by the interviewees to also consider attacker motivation (e.g. monetary gain, fame, destruction) and strategy (e.g. perform extensive monitoring before attacking) into the profiles. Implementing these suggestions will be a subject for future research.

A different line of future research is connected to broadening the supported attack description formalisms. Since attack trees have several known limitations, a richer formalism may be needed at some point of the development. One possible alternative to attack trees are timed automata (The TRESPASS Project, D3.3.2, 2015). The above-described basic set of parameters (budget, skill, time) still has meaning for automata and the respective implementation of the solution space pruning has rather a straightforward interpretation.

Still, more distinction is needed between continuous and categorical parameters. The continuous budget parameter can be modelled using the reward mechanism of the automata. The categorical parameters of skill and time may be used to reduce the model by removing the elementary attacks that do not satisfy the respective restriction (e.g. require the attacker to have a higher technical skill level than he has).

In addition, other parameters may be considered, with different functions in the analysis, in particular influencing attacker strategy. For example, motivation and knowledge would be such parameters. Considering motivation, different assets (outcomes) would have different utility values for different types of attackers: a cyber terrorist would be interested in causing damage (high utility), whereas a cyber spy would be interested in secret designs (high utility). “Tagging” attacker profiles and assets to be able to express such properties is a topic for further study.

In addition, knowledge of the attacker about the system may influence the attacker strategy. Without explicit attacker knowledge, analyses are typically “white-box”, assuming that the attacker knows the system. If we lift this assumption, there are different ways to account for limited knowledge. Firstly, separate actions could be introduced for gaining knowledge, with associated cost for the attacker. This would require being able to represent knowledge of the system explicitly as an outcome of an action. Alternatively, knowledge could be represented as an abstract parameter (Pieters & Davarynejad, 2015),

where less knowledge makes it more likely that the attacker chooses suboptimal attack strategies.

5.4. Including motivation

The above methods typically focus on attacker capabilities as a limiting factor in attack possibilities, thereby influencing risk. What such methods do not take into account is the motivation of the attacker. Rather than influencing the attack steps, the motivation of attackers influences the value of outcomes to the attacker, and thereby the goals.

In a master thesis supervised by Wolter Pieters and Jan van den Berg (Van Holsteijn, 2015), the motivation is assumed to have an influence on the pay-off an attacker receives from performing an attack. The text below is based on the management summary.

The value that including the motivation in attack tree analysis can bring may be summarised as the following:

- The gains parameter is made independent of the type of attacker;
- Various pay-offs are possible for variously motivated attackers;
- The gains parameter is made more realistic.

The framework is ensured to reach this potential added value, by adhering to a list of requirements. This list of requirements is build up from constraints to which the framework must conform and dilemmas for which a design choice has to be made. The resulting framework is mainly based on the method presented by (Lenin et al., 2014). Changes to the current method are mainly made to the gains parameter. The gains is no longer a global parameter that is only received by the attacker when reaching the root node. Instead it is possible to include intermediate pay-offs, which means that gains can also be allocated to intermediate nodes. In this way, different gains are possible for different attack paths in the attack tree. The gains can thus be represented in a more fine grained way. Also an opt-out possibility is included to allow attackers to perform attacks to only reach an intermediate node and not the root node of the attack tree.

In the current method the pay-off for an attacker was considered to be equal to the gains, which was the same for every type of attacker. This gains was also a single value. In the designed framework the gains has been slit up in five types of gains to deal with the five forms of motivation that an attacker may have, which are financial benefits, causing damage, knowledge gaining, pleasure seeking and gaining notoriety within a community. With the use of weight values, the importance of the various types of gains for an attacker can be represented. By multiplying the gains with the weight values, a pay-off can be calculated for a certain type of attacker. This way various pay-offs are possible for variously motivated attackers. Details of the calculations can be found in (Van Holsteijn, 2015).

A case study has been described to show the working of the framework on a real world case, which also served as a validation of the framework. In addition an expert opinion has been asked to validate the framework. The main improvement that can be made to

the framework by future research is focused on allocating values to the different types of gains and allocating the weight values for the different types of gains. Also attention could be paid to several dependencies between attack and attacker properties that have not yet been taken into account.

5.5. The library

The current attacker profile library (APriL) supports the following parameters:

- skill;
- budget;
- time.

For more detailed information about the parameters and how they are applied to the attack tree we refer the reader to Section 4.4. *Multi-Parameter Computations with Attacker Profiling* of The TRE$_S$PASS Project, D3.3.1 (2013). We expect to be able to add motivation in the final stages of the project.

TARA profiles and corresponding parameters will be made available for selection in the user interface.

5.6. Conclusions

Attacker profiles form the basis of the possibility to reuse the same system model for risk assessment under different threat contexts. We have described how attacker profiles can be used to inform risk analysis based on attack trees, including both capabilities and motivation of the expected attacker types. Separating the threat context from the system is one of the key innovations in TRE$_S$PASS. The attacker profiles can be shared in a library such that they can be reused for the analysis of other models. In the next chapter, we will look into sharing parts of the scenarios generated from models plus attacker profiles, in the form of attack patterns.

6. Attack Pattern Library

6.1. Strategy

The automatic generation of attacks from socio-technical security models depends on the level of detail available in the model. For example, to extract a file from a virtual machine on a certain hypervisor running on a certain kind of operating system, a very specific attack may be known and applicable. This detailed information, however, does clutter the model significantly, and will often not be available. The TRE$_S$PASS project has therefore devised an attack pattern library (APL) that is very similar in function and goal to the model pattern library and the attacker profiles.

Sharing attack pattern libraries is clearly of great benefit as it reduces the amount of modelling necessary at the individual organisation. At the same time, the attack pattern library, just like the other libraries discussed in this deliverable, contains potentially sensitive information, that should *not* be freely shared. In this chapter we discuss the strategy how the attack pattern library is used, to enable users of the TRE$_S$PASS tools and processes to make informed decisions about whether or not to share such information.

6.2. Functions of the APL

The attack pattern library serves two functions: it reduces clutter in the model representing the organisation, and it extends identified attacks with specific attack patterns. The attack pattern library differentiates between “standard” attack patterns and “domain-specific” attack patterns. Standard attack patterns represent general attacks, for example, opening a door by picking the lock, cracking it open, or tail gating an employee. Domain-specific patterns, on the other hand, are highly dependent on the organisation, and as such represent potentially dangerous information to share. An example are attacks exploiting specific knowledge about locations of keys in an organisation.

Neither the attack nor the attack patterns contain annotations about relevant properties of actions that form the input for the analyses performed on attacks. Instead, attacks and attack patterns contain holes that must be filled with the relevant numeric values.

The attack pattern library fulfils thus two functions:

- Once an attack has been identified, the attack is traversed, and its leaf nodes are inspected. If the leaf node has a label that matches with some known attack in the attack pattern rule, an attack expansion is triggered. To enable this, attacks

may contain leaf nodes with a label following some predefined rules, and the attack pattern library is then matched against this label based on information about the involved assets.

- On the leaf nodes, the attack pattern library performs property lookups, and decorates the leaf nodes, *e.g.*, with necessary resources, skill level, or risk of detection.

In general, the process described above applies whenever one or more agents or assets are involved in attack steps. The properties of agents and assets then contribute to the relevant parameters of the generated event, and influence the analysis results. For human actors in a system, properties may include nationality, age, awareness, etc. For assets, properties may include defence strength, protection class, or functional properties such as encryption. In The TRESPASS Project, D2.3.1 (2014) we have given examples of how information in the Attack Pattern Library and information on the agents involved are combined to obtain the parameters of the generated actions. We consider here two similar examples: an attacker trying to crack a door, and an attacker trying to social engineer a receptionist. In both cases the chance of success, the required time, and required skill level, amongst others, depend both on properties of the attacker and the password and receptionist.

Example 1

For cracking a door, the following sequence of analysis steps takes place:

1. The attack contains a leaf node of the attacker moving to the door.
2. The analysis retrieves the corresponding attack pattern from the library, which indicates that the parameter “time required” depends on the door’s protection class and the skill of the attacker, including a look-up table for the associated variables.
3. The analysis inspects all leafs in the pattern.
4. The analysis observes that the skill in the attacker profile is “very high”.
5. For picking the lock, the analysis observes that the door’s lock is “unpickable”.
6. For picking the lock, the analysis uses the look-up table to determine that the associated time required is “hours”.
7. The leaf node for picking the lock is annotated with this time.
8. For cracking the door, the analysis observes that the door’s protection class is “high”.
9. For cracking the door, the analysis uses the look-up table to determine that the associated time required is “minutes”.
10. The leaf node for cracking the door is annotated with this time.

Example 2

For social engineering a receptionist, the following sequence of analysis steps takes place:

1. The attack contains a leaf node where the attacker social engineers the receptionist to obtain access, in order to pass the door.
2. The analysis retrieves the corresponding attack pattern from the library, which indicates that the parameter “likelihood of success” depends on the age of the victim and the skill of the attacker, including a regression equation for the associated variables.
3. The analysis observes that the age of the receptionist is “25”.
4. The analysis observes that the skill in the attacker profile is “very high”.
5. The analysis uses the regression equation to determine that the likelihood of success is “0.7”.
6. The leaf node is annotated with this likelihood.

6.3. APL workflow

The Attack Pattern Library workflow is a part of the general TRE$_S$PASS workflow as summarised in The TRE$_S$PASS Project, D6.2.2 (2015).

As described above, the APL works as a mediator between the automated attack tree generation component developed in Work Package 3 and the various analysis tools developed in that work package. The function of the APL is to take generic attack trees as input and to extend it with more elaborate attack descriptions and to decorate these descriptions with parameter values. Accordingly, there are two components that are involved in the APL invocation:

1. First, the leafs of the generated tree are traversed and the leaf labels are matched to the list of actions (see The TRE$_S$PASS Project, D3.4.1 (2014)). Every action will have a basic match among the attack patterns, which is then used to extend the automatically generated tree.
2. Second, the attack including the added patterns must be annotated with data from the knowledge base generated from the system model and gathered social data. The annotated, expanded attack contains the parameter values and optionally also model-specific extensions, and is now suitable for analysis.

In the following we assume that attacks are represented as attack trees. This is no real restriction, but eases the description of the expansion and annotation of attacks.

```
label match {
  case IN attacker item container:
     // get type attacker from attacker profile
     // get type item from knowledge base
     // get type container from knowledge base
     // insert APL attacks that allow to extract item from container
  case MAKE attacker actor action:
     // get type attacker from attacker profile
     // get type actor from attacker profile
     // insert APL attacks based on types and action
  //...
}
```

Figure 6.1.: Code for the expansion of general attack trees in a context-unaware fashion. The expansion algorithm iterates over all leaf nodes and matches leaf node labels against the know cases. If a leaf node label matches a pattern in the attack pattern library, it is inserted into the general attack tree. Figure 6.3 illustrates this process.

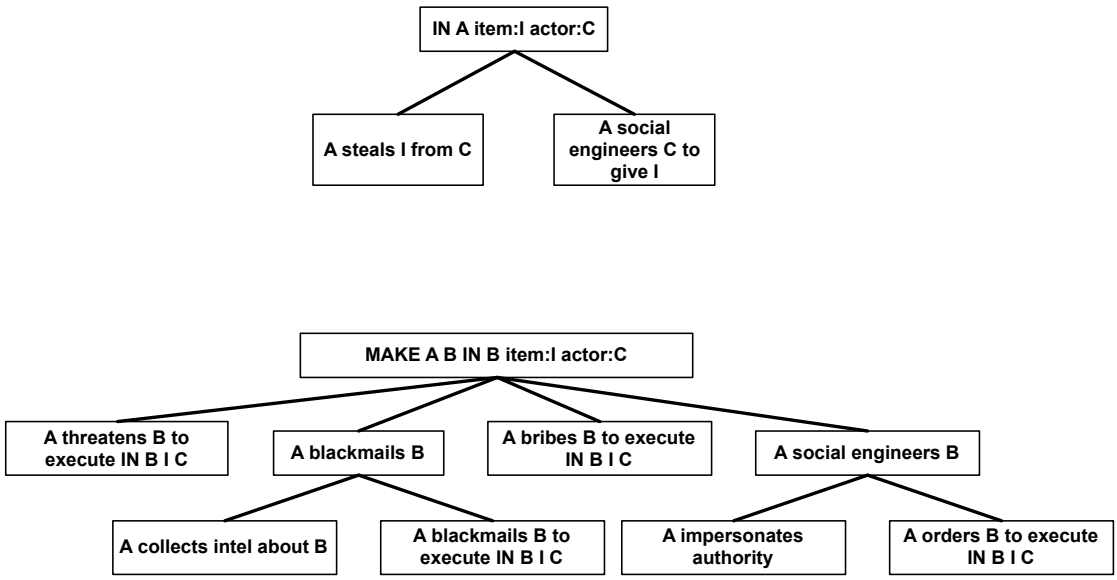

Figure 6.2.: Excerpts for two attack patterns. The upper one replaces inputing an item from an actor, and the lower one replaces social engineering an actor to input an item from another actor.

6.3.1. Attack Expansion

For expanding attacks with patterns from the APL, two approaches can be applied: either, one visits all the leaf nodes of the attack tree and deals with them in isolation, or one

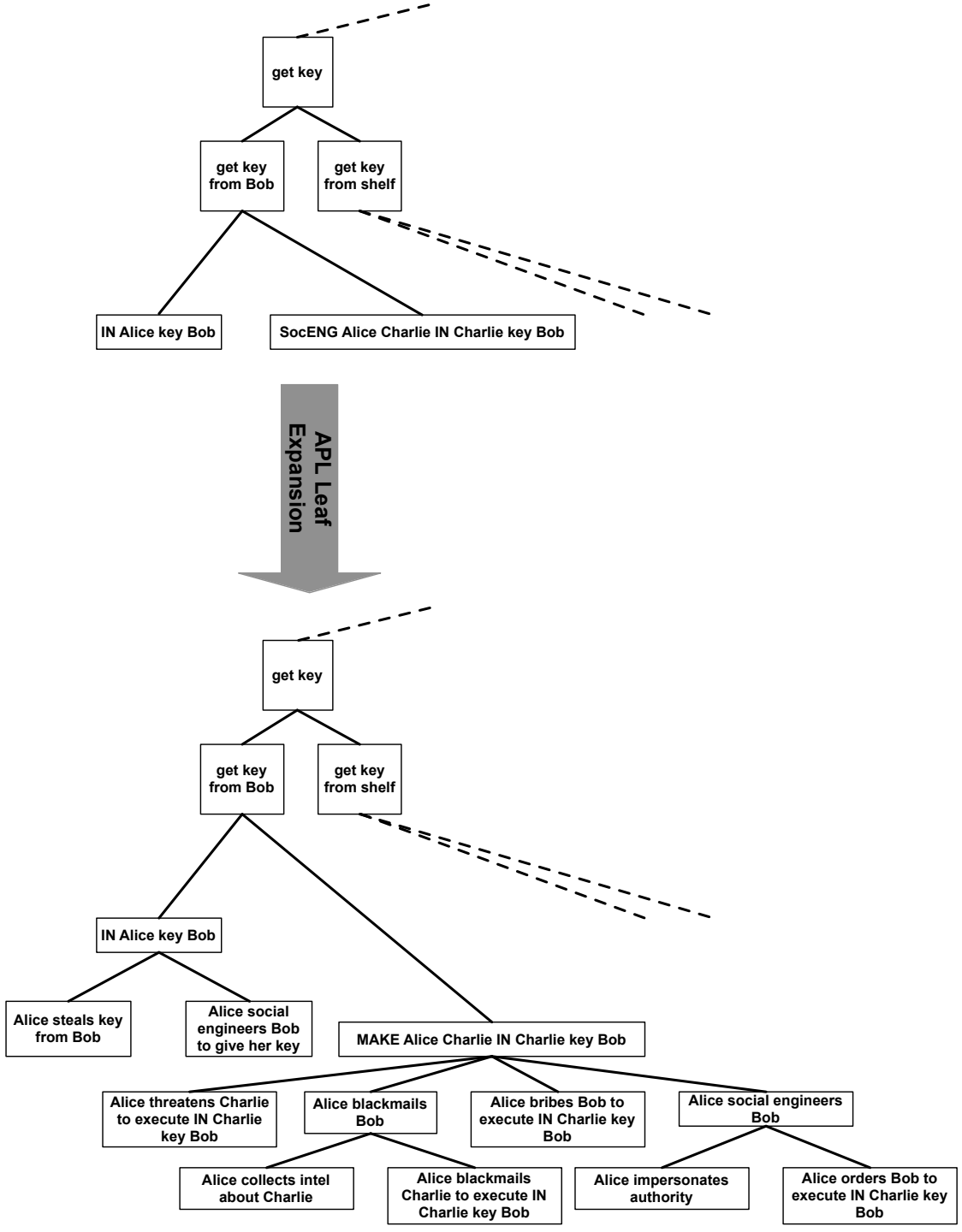

Figure 6.3.: The expansion of a part of a general attack tree. The patterns (Figure 6.2) may have holes, which are filled with attributes from the leaf node that is expanded.

performs a depth-first traversal of the attack tree and performs a context-aware analysis in the style of attribute grammars. The main difference is the added context in the second case.

In either approach, we are only interested in the leaf nodes, and only in those leaf nodes that represent a potential expansion node. As described above, in the TRE$_S$PASS workflow, these nodes are identified with labels that follow a certain grammar that describes input, output, social engineering, and forcing. Each of these basic actions takes an actor and acted on item or actor as arguments. Together, this information is used to expand the leaf node with more complicated patterns.

Figures 6.1, 6.2, and 6.3 show the pseudo code for matching labels, two example patterns, and an attack tree before and after the expansion, respectively. It should be clear how the patterns contribute to enhancing the generated attack tree with "standard" attack steps, that ideally should be shared.

6.3.2. Attack Annotation

The final step before handing attack trees to the analyses and visualisations is to annotate basic actions and elements of leaf nodes in the trees with values such as the cost of actions, the likelihood of success of an action, the time required to perform an action, etc. Just like the attack patterns described above, the information about these numerical attributes can in principle be shared between parties.

However, just like attack patterns, one must differentiate between specialised data, that is closely tied to an organisation and its employees, and general data, that is freely available. Examples for the former are specific data based on personnel records, examples for the latter is data based on general attributes such as gender, nationality, etc.

It should be noted that the annotations can be generated at the same time as the patterns are applied to the generated attack tree. However, while patterns are only applied to those leaf nodes that have matching labels, annotations must be applied to all leaf nodes. It may therefore be preferable to add these annotations in a second phase over all leaf nodes.

6.4. The library

The database of the APL contains attack trees with annotations. It is possible to break it into two databases – trees in one and annotations in another one. Data is stored in XML format. The trees part (without annotations) is currently implemented using Python and regular expressions. Depending on the next steps in attack generation (task 3.4), this prototype may need to be extended. The annotations part is planned for the final project year. We will provide an initial set of attack patterns for the case studies. TRE$_S$PASS consultants and users can then extend the library further. We expect that the annotations will be more case-specific than the trees themselves.

6.5. Conclusions

The attack pattern library provides an abstraction mechanism over attacks that moves technical and operational details of an attack from the model into the attack pattern library. This allows also for more robust and modular models, since attack patterns can be re-used within a model and even across different models. Using the attack pattern libray, both common attack patterns and domain specific attack patterns can be maintained and updated centrally and shared either within an organisation or in a wider context.

7. Attack Tree Sharing

7.1. Strategy

Apart from sharing attack patterns to extend generated attack trees, TRE$_S$PASS has also taken the initiative for setting up sharing of entire attack trees. This facilitates sharing of attack scenarios next to models (navigator maps), attacker profiles, and patterns.

Attack trees are the basic attack scenario model used throughout the TRE$_S$PASS process. Production of attack trees (be it automatic generation or manual tree design by an expert) is a complex task. In the same time, many attack trees, especially those manually designed by a security expert, are not specific to a system under question and can be reused in later analyses or by another organisation. Therefore, *attack tree sharing* is a useful practical instantiation of the model sharing practices reviewed in this deliverable and adopted in TRE$_S$PASS.

In this chapter we present an overview of an attack tree library set up (Section 7.2). We also present an approach to perform attack tree synthesis based on syntactic subtree selection and reuse (Section 7.3).

7.2. Open Attack Tree Library

The University of Luxembourg, supported by the TRE$_S$PASS partners, intends to set up a publicly available repository of attack trees and attack-defence trees that will enable simple sharing and reuse of trees in the research community.

7.2.1. Requirements on Attack Tree Library

Below we list a set of functional requirements that are applicable to an attack tree sharing service.

- **Open submission of new trees and access to available trees:** This is a basic functionality of an attack tree library.
- **Standard format of the trees:** To enable reuse the available trees need to conform to a single standard format. This format can be defined as an XML schema (e.g., the XML schema used by the ADTool (B. Kordy, Kordy, Mauw, & Schweitzer, 2013)), an attack tree meta-model, or in some other way.

- **Tree meta-data:** The library users might want to include some meta-data with the tree files, where they could specify the tree design details (authors' names, manual or automatic design, tools used, date of creation, application domain, comments, etc.).
- **Tree visualisation capabilities:** The users should have a way to visualise the available trees in order to be able to select among them.
- **Tree matching:** The library can support identification of similar trees based on syntactic or semantic similarities among them.
- **Search over trees and subtrees:** The library can support queries over available trees in order to fetch the trees relevant to the user request.
- **Private tree sharing:** The library can have an access control mechanism, similar to the access control mechanisms used on social networks, that will allow to share some sensitive trees only with trusted users.

7.2.2. Current Set Up

Reaching maturity of project results, we expect that TRE$_S$PASS will produce a large variety of automatically generated attack trees, that will be subsequently pruned, extended and decorated with attributes. These trees will be shared within the project to streamline collaboration, and, at later stages, outside the project. To enable sharing of these models we have set up an Attack Tree Library, currently in the format of git repository intended only for the TRE$_S$PASS partners[1]. Currently this repository satisfies the following requirements:

- **Open submission of new trees and access to available trees:** Any user with account on gitlab can access the available trees and submit new models to the repository.
- **Standard format of the trees:** Currently the ADTool XML schema is used (presented in the next section).
- **Tree meta-data:** Only as a part of XML file itself.
- **Tree visualisation capabilities:** Visualisation is available via the ADTool.

Other functional requirements are currently not supported.

[1] https://gitlab.com/twice3/Attack-Tree-Library.git

7.2.3. Tree Sharing Format

In this section we present the ADTool XML schema (in XSD format) that is used in the tree sharing library. This schema was selected as the initial tree sharing format because the submitted trees compliant with the schema can be easily visualised in ADTool. At later stages, different additional formats can be introduced, together with translation mechanisms, that will enable better tree diversity (for example, trees with sequential-AND refinements).

Figure 7.1 depicts the XML schema specifying attack-defence trees (ADTs) and attack trees with attribute values. The information in the remainder of this section is an excerpt from the ADTool manual (P. Kordy & Schweitzer, 2013).

Below we explain the elements, attributes, and types declared by this schema.

- The adtree element contains exactly one root node (line 5) and may contain a finite number of "domain" elements (line 6)
- Each node of an ADTree is represented by a "node" element (lines $10 - 18$), having
  - exactly one sub-element "label", corresponding to the label of the node,
  - a finite number of "parameter" sub-elements, corresponding to quantitative or qualitative measures and their values, and
  - a finite number of "node" sub-elements, corresponding to the node's children, including countering nodes.
- There are two attributes that characterise "node" elements: the "refinement" attribute and the "switchRole" attribute.
  - The "refinement" attribute represents the refinement of the node in the ADTree. This attribute has two possible values: "disjunctive" and "conjunctive" (lines $36 - 41$). The default value of the "refinement" attribute is "disjunctive", thus only conjunctive nodes require an explicit specification of the value for the "refinement" attribute.
  - The "switchRole" attribute is used to switch between an attacker and a defender role. This is a Boolean attribute with two possible values "yes" and "no" (lines $43 - 48$). When "switchRole" is set to " yes", the node corresponding to the current element belongs to the opposite player than its parent node.
- The values of element "label" (lines $20 - 25$) are strings that can be composed of the following characters: numbers from 0 to 9, capital and small letters of the English alphabet, space, horizontal tab, enter, ?, !, -, _, ', semicolon, colon, #, [, {, },], =, *, /, \, |, @, ^, ‘, ", &, $, ~, ., < ,>, +, %.
- The "parameter" element contains the value of the quantitative or qualitative measure that it represents (lines $27 - 34$).
- There are two attributes that characterize the "parameter" elements: "domainId" attribute (line 30) and "category" attribute (line 31)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="adtreeType">
    <xsd:sequence>
      <xsd:element name="node" type="nodeType" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="domain" type="domainType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="nodeType">
    <xsd:sequence>
      <xsd:element name="label" type="labelType" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="parameter" type="parameterType" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="node" type="nodeType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="refinement" type="refinementType" use="optional" default="disjunctive"/>
    <xsd:attribute name="switchRole" type="booleanType" use="optional" default="no"/>
  </xsd:complexType>

  <xsd:simpleType name="labelType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[0−9A−Za−z\s\?!\−_':;#\[\{\}\]=\*/\\\|@\^`&amp;quot;&amp;amp;$~\.&amp;lt;&amp;gt;\+%]+"/>
      <xsd:whiteSpace value="preserve"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="parameterType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="domainId" type="xsd:string" use="required"/>
        <xsd:attribute name="category" type="categoryType" use="optional" default="basic"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

  <xsd:simpleType name="refinementType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="disjunctive"/>
      <xsd:enumeration value="conjunctive"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="booleanType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="yes"/>
      <xsd:enumeration value="no"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="categoryType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="basic"/>
      <xsd:enumeration value="default"/>
      <xsd:enumeration value="derived"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="domainType">
    <xsd:sequence>
      <xsd:element name="class" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="range" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="tool" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
  </xsd:complexType>

  <xsd:element name="adtree" type="adtreeType">
    <xsd:key name="DomainKey">
     <xsd:selector xpath=".//domain"/>
     <xsd:field xpath="@id"/>
    </xsd:key>
    <xsd:keyref name="DomainKeyRef" refer="DomainKey">
     <xsd:selector xpath=".//parameter"/>
     <xsd:field xpath="@domainId"/>
    </xsd:keyref>
  </xsd:element>
</xsd:schema>
```

Figure 7.1.: XSD schema adtree.xsd specifying XML files

– The (required) "domainId" attribute references to the corresponding " domain" element

  - The (optional) "category" attribute (lines $50 - 56$) specifies
    - which values can be edited (category="basic"), these are values of the non-refined nodes that can be modified by the user,
    - which are non-editable default values (category="default"), these are basic values assigned to non-refined nodes that cannot be changed by the user, and
    - which values are results of a computation (category="derived"), these are values computed at the refined nodes.
- Each "domain" element (lines $58 - 65$) contains exactly one "class" sub-element, zero or one "range" sub-element, and a finite number of "tool" sub-elements.
  - The "class" sub-element represents the name of the Java class that implements the corresponding domain (line 60).
  - In the case when the domain is restricted (e.g., Reachability of the proponent's goal in less than k units), the sub-element "range" (line 61) stores the value of the used restriction parameter (i.e., k).
  - The "tool" elements (line 62) list the tools in which the current domain is used. The ADTool ignores the content of all "domain" elements which do not contain the "tool" sub-element having value 'ADTool'.
- The attribute called "id" uniquely identifies each "domain" element (line 64).
- Lines $67 - 71$ ensure that each "domain" element has a unique identifier "id".
- Lines $72 - 76$ guarantee that the "domainId" attribute of the "parameter" elements references to one of the declared domains.

7.3. Synthesis of Attack-Defence Trees using a Library

Attack-defence trees are a formalism and a graphical security model that aim at combining the attacker's and the defender's views on achieving some security goal for a given system. They visualise, for example, potential attacks on the system in question and capture security controls that are available to mitigate these attacks. In the rest of this section we assume familiarity of the reader with attack-defence trees. The interested reader can refer to B. Kordy, Mauw, Radomirović, and Schweitzer (2012) for more details on this formalism. Notice that the attack-defence tree formalism subsumes the attack tree formalism (in the sense that any attack tree can be interpreted as an attack-defence tree with no defence nodes). Therefore the approach to attack-defence tree synthesis proposed below is also directly applicable to attack trees.

Typically, attack-defence trees are designed manually by a security expert. Therefore, such trees are as representative of actually occurring attacks as the expert's knowledge is representative of the real security situation. Creation of comprehensive attack-defence

trees is usually a time-consuming process. In the same time, the TRESPASS project is among the pioneers of automated attack generation from the socio-technical system models. The quality of such automatically generated trees, in turn, is tightly correlated with the quality of the socio-technical model designed to represent the system in question. Again, design of a comprehensive system model is a manual time-consuming process. Therefore, sharing and reuse of subtrees encapsulating detailed security knowledge of particular systems can be beneficial for both manually designed and automatically generated attack-defence trees.

Let us consider the following practical scenario. A security officer in a large organisation wants to identify all possible ways for an attacker to achieve a certain goal, e.g. "Compromise confidential client list". The security officer starts by creating his own tree A_1 of how to compromise the sensitive client list. Yet, because the organisation has a complex structure, he is only able to cover a part of the full tree. Another part A_2 will be done by the IT-department that will also consider the network topology and the relevant IT systems. By combining A_1 with A_2 and a public online library of ADTs, a more complete ADT can be created, including all well-known atomic attacks against the organisation's infrastructure. Because a public library can be constantly updated, the security officer saves time, and the end-result will be less error-prone, more accurate and updated than a manually refined tree.

The knowledge sharing in the form of attack-defence trees requires the users to be able to quickly navigate through some tree library and select relevant (sub)trees. To this extent, we can envisage that the end-user will be able to express his requirements on the relevant trees in some easy-to-use format, and the library will be able to look up the relevant trees. This format can be, for example, captured as a syntactic definition of a tree.

Notice that there is an alternative possibility to use the standard attack-defence tree semantics proposed in B. Kordy et al. (2012). However, we start with syntactic tree matching because we believe it is more straight-forward for the end-users to express their requirements in such format.

7.3.1. Syntactic Tree Definition Format

Each attack-defence tree (ADT for short) can be captured by a syntactic representation of its structure and nodes. We propose a hierarchical definition of trees expressed as their root nodes.

Node. A node A is a tuple $\langle n, R, E\rangle$, consisting of its Name $n \in \mathbb{N}$, a relation to sub-nodes $R \in \mathbb{R}$, and a collection of variables $E \in \mathbb{E}$ called *Environment*.

A domain of ADTs is formally defined as

$$\mathbb{A} := \mathbb{N} \times \mathbb{R} \times \mathbb{E} \tag{7.1}$$

Parameter	Value
Crime scene	Place d'Armes
Name of shop	McDonalds
Expected outcome	€1.000

Table 7.1.: Example of specification of parameters

Name The name $n \in \mathbb{N}$ is the identifier of the node. It consists of a label L, a type of node T and collection of parameters P.

The domain of names $\mathbb{N}$ can be formalised as follows:

$$\mathbb{N} := \mathbb{L} \times \mathbb{T} \times \mathbb{C} \tag{7.2}$$

Label $L \in \mathbb{L}$ is a sting in the natural language describing the attack or defence of this particular node, for example "steal money from bank" or "install security cameras".

Type $T \in \mathbb{T}$ determines whether the node is an attack node α, or a defence node β. According to the attack-defence tree formalism, any node can only have one sub-node of opposite type, whereas it can have an unlimited number of sub-nodes of same type.

Set of parameters $P \subseteq \mathbb{C}$ is a collection of local parameters. These are used to personalise nodes by introducing, e.g., attributes, such as time or cost. A parameter is not essential information to the structure of the ADT, but it is solely used for personalisation. Generic nodes, downloaded from a library, are able to work across organisations and situations, given the provided preconditions. By introducing parameters it is possible to personalise a node to a specific situation by including various relevant information.

A parameter $p \in P$ is a key-value pair $\langle \omega, \psi \rangle$, where the key ω specifies the relevant description, and ψ gives the value of this parameter.

P contains all parameters from the node itself, and also all parameters from all sub-nodes. An example set of parameters P is given in Table 7.1.

Relation. A relation $R \in \mathbb{R}$ consists of a collection of sub-nodes, if any, and a description of relation $D \in \mathbb{D}$ among the sub-nodes.

The domain $\mathbb{R}$ of relations is formally defined as:

$$\mathbb{R} := \mathbb{D} \times \mathbb{S}. \tag{7.3}$$

D determines whether a node is, e.g., a leaf node or it has sub-nodes in a specified order. D can take the following values:

- *Disjunctive* ($\vee$): an exploit of a single sub-node is enough to compromise the goal (an OR-refinement on children nodes).

- *Conjunctive* ($\wedge$): all sub-nodes have to be exploited before the parent node can be compromised (an AND-refinement on children nodes).
- *Sequential conjunctive* ($\overline{\wedge}$): all sub-nodes have to be exploited in a specified order before the parent node is compromised (a sequential-AND refinement on children nodes).
- *Reference* ($\rightarrow$): the node A is already refined in another node.
- *Leaf node* ($\perp$): the node has no sub-nodes.

As we are considering only tree structures, it is never possible to return to the same node twice, by following sub-nodes or references in one direction. A node can be included as a sub-node in multiple trees, as long as no cycles appear.

Environment. $E \subseteq \mathbb{E}$ is a collection of independent variables $V_i \subseteq \mathbb{V}$, which are preconditions to a node. A *precondition* is a circumstance which has to be fulfilled before the node can work. By comparing the preconditions of two nodes, it is possible to determine if one node is *compliant* to the other, hence if the nodes will be able to be part of the same tree. The notion of compliance is further explained in Section 7.3.3.

A variable V is a tuple $\langle \varphi, \rho, \lambda \rangle$, where φ specifies the category of the variable or precondition, e.g. "OS". There can be multiple instances of the same category within the same node. ρ specifies the value attached to the category, e.g. "Windows 8.1". Within each category there is exactly one instance of the same value. λ determines if the node applies to the category and attached value or not. For instance, a virtual attack described by a node may work on "Windows 8", but the same attack will not work on "Windows 8.1".

Each node may include multiple variables, which detail the environment in which the node is working. All variables in children nodes, also occur in their parent nodes, as demonstrated in Figure 7.2. Usually variables are introduced by children nodes, but it is possible to introduce new variables into top nodes directly, if it does not contradict any existing variable. For instance, is it possible to introduce a variable stating a node's compliance to "Linux Debian 8.1" into a parent node, unless there is a sub-node with a variable describing that the node is non compliant to "Linux Debian 8.1".

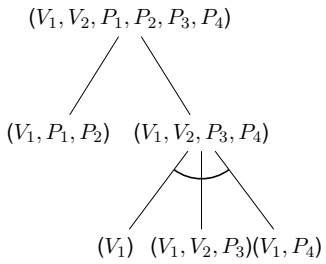

Figure 7.2.: Variables and parameters in sub-nodes are also occurring in top nodes

φ	ρ	λ
OS	Windows 7	Yes
OS	Windows 8	Yes
OS	Windows 8.1	No
Adobe Reader	V. 9.4.7	Yes

Table 7.2.: Example of specification of variables

Let us consider Table 7.2 as an example of variables in a tree. The variables specify that the tree is compliant to computers running “Windows 7” or “Windows 8” as OS, but not “Windows 8.1”. Furthermore “Adobe Reader v. 9.4.7” has to be installed.

The difference between the parameters P and environment E is that P is only known within the same ADT, whereas E is known among many ADTs. In essence, parameters represent a local description and personalisation of a tree that is not important for its reuse, while the environment describes the attack context essential to know for reuse.

7.3.2. Collection of ADTs

Attack-Defence Tree Forest. A set of ADTs sharing the same environment is called a *Forest*. Formally, a forest $\mathbb{F}$ is defined as

$$\mathbb{F} := \{A_1, \ldots, A_n\} \tag{7.4}$$

where trees $A_1, \ldots, A_n$ all work under the same environment E_F.

We can envisage that forests are created from large collections of independent trees and are used as a portfolio for different systems and environments. For example, one can create a forest to describe Java attacks, or Windows OS attacks.

Collections of Trees. We can also consider collections of independent ADTs that do not share environments. Opposite to the forest, there is no requirement of compliance among the trees in a collection (denoted $\mathbb{Q}$).

$$\mathbb{Q} := \{A_1, \ldots, A_n\} \tag{7.5}$$

where there is no restriction on $A_1, \ldots, A_n$ (the variables might be distinct to or in union with each other, etc.).

7.3.3. Compliance

To enable an automatic search of relevant ADTs from a collection, we need to define what is a *relevant* or *compliant* tree. The notion of compliance essentially describes whether an ADT A_1 will be able to work under the environment of A_2. We start by defining compliance of individual variables.

Definition 7.3.1. *Two variables V_1 and V_2 are compliant, iff*

φ_1	=	φ_2
ρ_1	=	ρ_2
λ_1	=	λ_2

Compliant variables are expressed as $V_1 = V_2$.

Definition 7.3.2. *Two variables V_1 and V_2 are not compliant, iff*

φ_1	=	φ_2
ρ_1	=	ρ_2
λ_1	$\neq$	λ_2

Non-compliant variables are expressed as $V_1 \neq V_2$.

Definition 7.3.3. *Two variables V_1 and V_2 are indeterminably compliant, iff*

φ_1	=	φ_2
ρ_1	$\neq$	ρ_2
λ_1	?	λ_2

or

φ_1	$\neq$	φ_2
ρ_1	?	ρ_2
λ_1	?	λ_2

where the relation "?" expresses that the relation among the values is subordinate. Indeterminably compliant variables are expressed as $V_1 \stackrel{?}{=} V_2$.

Given two arbitrary single variables, is it always possible to evaluate their compliance to each other using the definitions above.

Compliance of nodes. Usually a single node will have multiple preconditions to be fulfilled, hence it will contain an equal amount of variables. Two collections of variables, E_1 and E_2, can be evaluated for compliance of E_1 to E_2, hence the compliance of node A_1 to A_2. Just like a single variable, can E_1 be either *Compliant*, *Non compliant* or *Indeterminably compliant* to E_2. Opposite to a single variable, compliance of a collection of variables does not work bi-directionally. For instance, a node A_1 can be determined as compliant to node A_2, but that does not imply A_2 is compliant to A_1. A_2 may contain preconditions not considered by A_1, hence A_2 is indeterminably compliant to A_1.

Definition 7.3.4. *A_1 is fully compliant to A_2, if $\forall\ V_i \in E_1$ and $\exists\ V_j \in E_2$ such that $V_i = V_j$. This can also be expressed as $A_1 \subseteq A_2$.*

Definition 7.3.5. *A_1 is partially compliant to A_2, if $\forall\ V_i \in E_n$, where $E_n \subset E_1$, and $\exists\ V_j \in E_2$ such that $V_i = V_j$. This can also be expressed as $A_n \subseteq A_2$, where $A_n \subset A_1$.*

Definition 7.3.6. *A_1 is not compliant to A_2, if $\exists\ V_i \in E_1$ and $\exists\ V_j \in E_2$ such that $V_i \neq V_j$.*

Definition 7.3.7. *A_1 is indeterminably compliant to A_2, if $\exists\ V_i \in E_1$, $\forall\ V_j \in E_2$ such that $V_i \stackrel{?}{=} V_j$.*

Using the definitions above it is always possible to determine if an arbitrary node A_1 is compliant to another arbitrary node A_2.

7.3.4. Automated ADT extension

Given an ADT A_i and a large shared collection of ADTs $\mathbb{Q}$, the desired scenario is to automatically supply relevant subtrees from $\mathbb{Q}$ to A_i. In this way we can achieve more complete ADT, created in a time-saving semi-automatic manner.

Initially a tree A_0 is manually designed by the user, together with some description of required preconditions E_{A_0}. To make the automatic expansion more efficient, A_0 needs to be as complete as possible. The environment E_0 of A_0 should be defined as precisely as possible, e.g. by including circumstances, under which the node will not work. These preconditions stating non-compliance can be an effective way to limit the number of useful results returned from $\mathbb{Q}$.

The public collection of ADTs $\mathbb{Q}$ contains multiple trees, of which some are compliant to A_0 and some are not (thus they are not of interest to the user). Our automatic process of expanding the tree starts by extracting a subset forest F_0, compliant to A_0 from $\mathbb{Q}$ (steps 2 and 3 in Figure 7.3). Subsequently, all nodes of A_0 are looked up in F_0, and the relevant nodes are merged into A_{Merged} (steps 4 and 5 in Figure 7.3). The result A_{Merged} is a (potentially) very large ADT. If desired by the user, all references occurring in A_{Merged} can be replaced by complete trees, producing the final static tree A_{Final} with all branches refined until atomic attacks/defences (steps 6 and 7 in Figure 7.3).

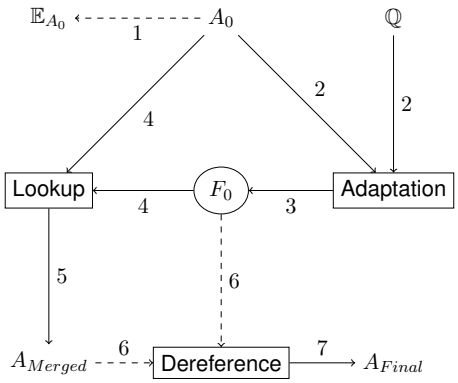

Figure 7.3.: Steps of operations

7.4. Conclusions

Next to the libraries for model patterns, attacker profiles and attack patterns, we believe that it is also useful to provide the possibility for sharing entire attack trees. This enables sharing of both hand-crafted attack trees and attack trees generated by the attack navigator tools. To this end, we have discussed the Open Attack Tree Library and associated data format, as well as the synthesis of attack-defence trees using this library.

8. Conclusions

This deliverable presents the state of development in Task 5.3 “Model creation, sharing, and maintenance” of the TRE$_S$PASS project as of the end of Month 36. The issue of model management has been approached from several viewpoints.

As the first contribution, various approaches to sharing security information have been studied. The study includes references to various relevant standards and a draft of upcoming NIST Guide to Cyber Threat Information Sharing.

Secondly, the deliverable described the best practices for model creation developed in the project. These range from formal languages to physical modelling and the use of argumentation. All of these provide opportunities to identify relevant modelling elements and integrate them into a full-fledged attack navigator map. Based on the model development practices, specific needs for sharing have been identified.

To enable this sharing, standard parts of the TRE$_S$PASS models will be stored in libraries. At the moment, three kinds of libraries have been envisioned. During the modelling phase, standard items will be needed – namely, attacker profiles describing the motivation and capabilities of relevant attackers, and model components describing various real-life artifacts (like doors, computers, chip cards, etc). In TRE$_S$PASS, such descriptions will be kept in the Attacker Profile Library (APriL) and the Model Pattern Library (MPL). The current deliverable presents an overview of these libraries and how they can be used. In addition, the Attack Pattern Library (APL) will store standard attack descriptions in the form of attack trees, attack-defence trees, automata or anything else that the computational routines require.

We plan to work on the following topics in year 4 (not exhaustive):

- Model maintenance (The TRE$_S$PASS Project, D5.3.3, 2016);
- Further integration of the data collection of WP2 with the attack pattern library;
- Further extending attacker profiles and model patterns;
- Publishing patterns for sharing;
- Processes for sharing as part of the integrated TRE$_S$PASS process, in relation to the discussion in chapter 2.

References

The 2013 information security breaches survey. (2013). Retrieved from https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/200455/bis-13-p184-2013-information-security-breaches-survey-technical-report.pdf

Alexander, C., Ishikawa, S., & Silverstein, M. (1977). *A pattern language: Towns, buildings, construction*. New York: Oxford University Press. Hardcover. Retrieved from http://www.amazon.fr/exec/obidos/ASIN/0195019199/citeulike04-21

Band, I., et al. (2015). *Modeling enterprise risk management and security with the ArchiMate language.* The Open Group White Paper W150.

Barnum, S., & Sethi, A. (2007). *Attack patterns as a knowledge resource for building secure software* (Tech. Rep.). Cigital Inc.

Bayley, I. (2014). Challenges for a formal framework for patterns. In C. Blackwell & H. Zhu (Eds.), *Cyberpatterns* (p. 47-55). Springer International Publishing. Retrieved from http://dx.doi.org/10.1007/978-3-319-04447-7_4 doi: 10.1007/978-3-319-04447-7_4

Blackwell, C. (2012). A Strategy for Formalizing Attack Patterns. In *The first international workshop on cyberpatterns unifying design patterns with security, attack and forensic patterns.* Retrieved from http://cms.brookes.ac.uk/staff/HongZhu/CyberPatterns2012/Cyberpatterns2012Proceedings.pdf#page=39

Casey, T. (2007). Threat agent library helps identify information security risks. *Intel White Paper, September*. Retrieved from https://communities.intel.com/docs/DOC-1151

Casey, T., Koeberl, P., & Vishik, C. (2010). Threat agents: A necessary component of threat analysis. In *Proceedings of the sixth annual workshop on cyber security and information intelligence research* (pp. 56:1–56:4). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1852666.1852728 doi: 10.1145/1852666.1852728

Cuppens, F., & Miège, A. (2002). Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy* (pp. 202–). Washington, DC, USA: IEEE Computer Society. Retrieved from http://dl.acm.org/citation.cfm?id=829514.830542

Fang, F., Parameswaran, M., Zhao, X., & Whinston, A. (2012). An economic mechanism to manage operational security risks for inter-organizational information systems. *Information Systems Frontiers*, 1-18. Retrieved from http://dx.doi.org/10.1007/s10796-012-9348-y doi: 10.1007/s10796-012-9348-y

Fernandez, E. B., Pelaez, J. C., & Larrondo-Petrie, M. M. (2007). Attack patterns: A new forensic and design tool. In P. Craiger & S. Shenoi (Eds.), *Ifip int. conf. digital*

forensics (Vol. 242, p. 345-357). Springer. Retrieved from http://dblp.uni-trier.de/db/conf/ifip11-9/df2007.html#FernandezPL07

Frincke, D., Tobin, D., Mcconnell, J., Marconi, J., & Polla, D. (1998). A framework for cooperative intrusion detection. In *Proc. 21st NIST-NCSC National Information Systems Security Conference* (pp. 361–373).

Gal-Or, E., & Ghose, A. (2005). The economic incentives for sharing security information. *Info. Sys. Research*, *16*(2), 186–208. Retrieved from http://dx.doi.org/10.1287/isre.1050.0053 doi: 10.1287/isre.1050.0053

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Gao, X., Zhong, W., & Mei, S. (2013). Security investment and information sharing under an alternative security breach probability function. *Information Systems Frontiers*, 1-16. Retrieved from http://dx.doi.org/10.1007/s10796-013-9411-3 doi: 10.1007/s10796-013-9411-3

Gegick, M., & Williams, L. (2007, April). On the design of more secure software-intensive systems by use of attack patterns. *Inf. Softw. Technol.*, *49*(4), 381–397. Retrieved from http://dx.doi.org/10.1016/j.infsof.2006.06.002 doi: 10.1016/j.infsof.2006.06.002

Gkantsidis, C., & Rodriguez, P. (2006). Cooperative security for network coding file distribution. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings* (p. 1-13). doi: 10.1109/INFOCOM.2006.233

Heath, C. H. P., Coles-Kemp, L., Hall, P. A., et al. (2014). Logical lego? co-constructed perspectives on service design. In *Ds 81: Proceedings of norddesign 2014, espoo, finland 27-29th august 2014.*

Huang, Y.-a., & Lee, W. (2003). A cooperative intrusion detection system for ad hoc networks. In *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks* (pp. 135–147). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/986858.986877 doi: 10.1145/986858.986877

Ionita, D., Wieringa, R., Bullee, J.-W., & Vasenev, A. (2015, October). *Tangible modelling to elicit domain knowledge: an experiment and focus group.* Springer. (To be published)

ISO/IEC/IEEE. (2011). *42010-2011 - systems and software engineering — architecture description.*

Johnson, C., Badger, L., & Waltermire, D. (2014, October). *NIST special publication 800-150 (draft) guide to cyber threat information sharing* (Tech. Rep.). National Institute of Standards and Technology.

Jurjens, J. (2005). *Secure systems development with UML*. Springer.

Kordy, B., Kordy, P., Mauw, S., & Schweitzer, P. (2013). Adtool: security analysis with attack–defense trees. In *Quantitative evaluation of systems* (pp. 173–176). Springer.

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

Kordy, P., & Schweitzer, P. (2013). *Adtool manual, http://satoss.uni.lu/members/piotr/adtool/manual.pdf.*

Koutepas, G., Stamatelopoulos, F., & Maglaris, B. (2004). Distributed management architecture for cooperative detection and reaction to DDoS attacks. *J. Netw. Syst. Manage.*, *12*(1), 73–94. Retrieved from http://dx.doi.org/10.1023/B:JONS.0000015699.50210.e3 doi: 10.1023/B:JONS.0000015699.50210.e3

Lenin, A., Willemson, J., & Sari, D. P. (2014). Attacker Profiling in Quantitative Security Assessment Based on Attack Trees. In *NordSec'14: Proceedings of the 19th Nordic Conference on Secure IT Systems.* Springer. (to appear)

Locasto, M., Parekh, J., Keromytis, A., & Stolfo, S. (2005). Towards collaborative security and P2P intrusion detection. In *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC* (p. 333-339). doi: 10.1109/IAW.2005.1495971

Lund, M. S., Solhaug, B., & Stølen, K. (2011). *Model-driven risk analysis: the CORAS approach*. Springer Berlin Heidelberg. Retrieved from http://www.springer.com/computer/swe/book/978-3-642-12322-1

Mateski, M., Trevino, C. M., Veitch, C. K., Michalski, J., Harris, J. M., Maruoka, S., & Frye, J. (2012). *Cyber threat metrics* (Tech. Rep. No. SAND2012-2427). Albuquerque, New Mexico 87185: Sandia National Laboratories.

Moore, A. P., Ellison, R. J., & Linger, R. C. (2001). *Attack Modeling for Information Security and Survivability* (Technical Note No. CMU/SEI-2001-TN-001). Carnegie Mellon University.

Narasimhan, H., Varadarajan, V., & Chandrasekaran, P. R. (2010). Towards a cooperative defense model against network security attacks. In *WEIS.*

Nojiri, D., Rowe, J., & Levitt, K. (2003). Cooperative response strategies for large scale attack mitigation. In *DARPA information survivability conference and exposition, 2003. proceedings* (Vol. 1, p. 293-302 vol.1). doi: 10.1109/DISCEX.2003.1194893

Peter Hall, L. C.-K., Claude Heath, & Tanner, A. (2015). Examining the contribution of critical visualisation to information security. In *Proceedings of the 2015 new security paradigms workshop.* ACM.

Pieters, W., & Davarynejad, M. (2015). Calculating adversarial risk from attack trees: Control strength and probabilistic attackers. In J. Garcia-Alfaro et al. (Eds.), *Data privacy management, autonomous spontaneous security, and security assurance* (Vol. 8872, p. 201-215). Springer International Publishing. Retrieved from http://dx.doi.org/10.1007/978-3-319-17016-9_13 doi: 10.1007/978-3-319-17016-9_13

Pieters, W., Hadžiosmanović, D., Lenin, A., Montoya, L., & Willemson, J. (2014). Poster Abstract: TRE$_S$PASS: Plug-and-Play Attacker Profiles for Security Risk Analysis. In *Proceedings of the 35th IEEE Symposium on Security and Privacy.* Retrieved from http://www.ieee-security.org/TC/SP2014/posters/PIETE.pdf (Poster and Extended Abstract)

Pushpakumar, H. (2015). *Understanding the threat landscape in e-government infrastructure for business enterprises* (Master's thesis). Retrieved from http://repository.tudelft.nl/view/ir/uuid%3A5b21c91a-2f64-4117-9ad8-c9487e92b72b/

Rosenquist, M. (2009). *Prioritizing Information Security Risks with Threat Agent Risk Assessment.* Whitepaper.

Shirazi, N.-u.-h., Schaeffer-Filho, A., & Hutchison, D. (2014). Attack pattern recognition through correlating cyber situational awareness in computer networks. In C. Blackwell & H. Zhu (Eds.), *Cyberpatterns* (p. 125-134). Springer International Publish-

ing. Retrieved from http://dx.doi.org/10.1007/978-3-319-04447-7_10 doi: 10.1007/978-3-319-04447-7_10

The Open Group. (2013). *ArchiMate 2.1 specification*. Van Haren Publishing.

The TRESPASS Project, D1.1.2. (2015). *Final specifications and requirements for socio-technical security models.* (Deliverable D1.1.2)

The TRESPASS Project, D1.3.1. (2013). *Initial prototype of the socio-technical security model.* (Deliverable D1.3.1)

The TRESPASS Project, D1.3.2. (2015). *Extensibility of socio-technical security models.* (Deliverable D1.3.2)

The TRESPASS Project, D1.3.3. (2015). *Dynamic features of socio-technical security models.* (Deliverable D1.3.3)

The TRESPASS Project, D2.3.1. (2014). *Social data and policy extraction prototype.* (Deliverable D2.3.1)

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

The TRESPASS Project, D3.3.2. (2015). *TRESPASS methods for stochastic analysis.* (Deliverable D3.3.2)

The TRESPASS Project, D3.4.1. (2014). *Attack generation from socio-technical security models.* (Deliverable D3.4.1)

The TRESPASS Project, D5.1.2. (2015). *Final requirements for process integration.* (Deliverable D5.1.2)

The TRESPASS Project, D5.3.1. (2013). *Abstraction levels for model sharing.* (Deliverable D5.3.1)

The TRESPASS Project, D5.3.3. (2016). *Best practices for model maintenance.* (Deliverable D5.3.3)

The TRESPASS Project, D5.4.1. (2015). *First report on the integrated TRESPASS process.* (Deliverable D5.4.1)

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

The TRESPASS Project, D6.3.1. (2015). *TRESPASS user interface.* (Deliverable D6.3.1)

Uzunov, A. V., & Fernandez, E. B. (2014). An extensible pattern-based library and taxonomy of security threats for distributed systems. *Computer Standards & Interfaces*, *36*(4), 734 - 747.

van der Wagen, W., & Pieters, W. (2015). From cybercrime to cyborg crime: Botnets as hybrid criminal actor-networks. *British Journal of Criminology*, *55*(3), 578-595. Retrieved from http://bjc.oxfordjournals.org/content/55/3/578.abstract doi: 10.1093/bjc/azv009

Van Holsteijn, R. (2015). *The motivation of attackers in attack tree analysis* (Master's thesis). Retrieved from http://repository.tudelft.nl/view/ir/uuid%3A41044ff2-3aaf-49e9-bcf5-639bbd36d800/

Verbij, R. (2014). *Dutch e-voting opportunities: Risk assessment framework based on attacker resources* (Master's thesis, University of Twente). Retrieved from http://essay.utwente.nl/65811/

Zhu, Y. (2015). *Attack pattern ontology: A common language for attack information sharing between organizations* (Master's thesis). Retrieved from http://repository.tudelft.nl/view/ir/uuid%3A611583f1-b200-4851-915e-76a43c42fd46/

A. Project Summary

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

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

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

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

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

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

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

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

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

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

A.1. Case Studies

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

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

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

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

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

A.2. Overview of TRESPASS Integration

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

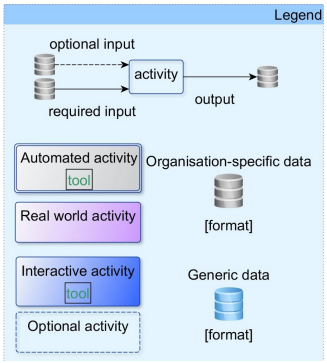

Figure A.1.: Legend for the Integration diagram in Figure A.2.

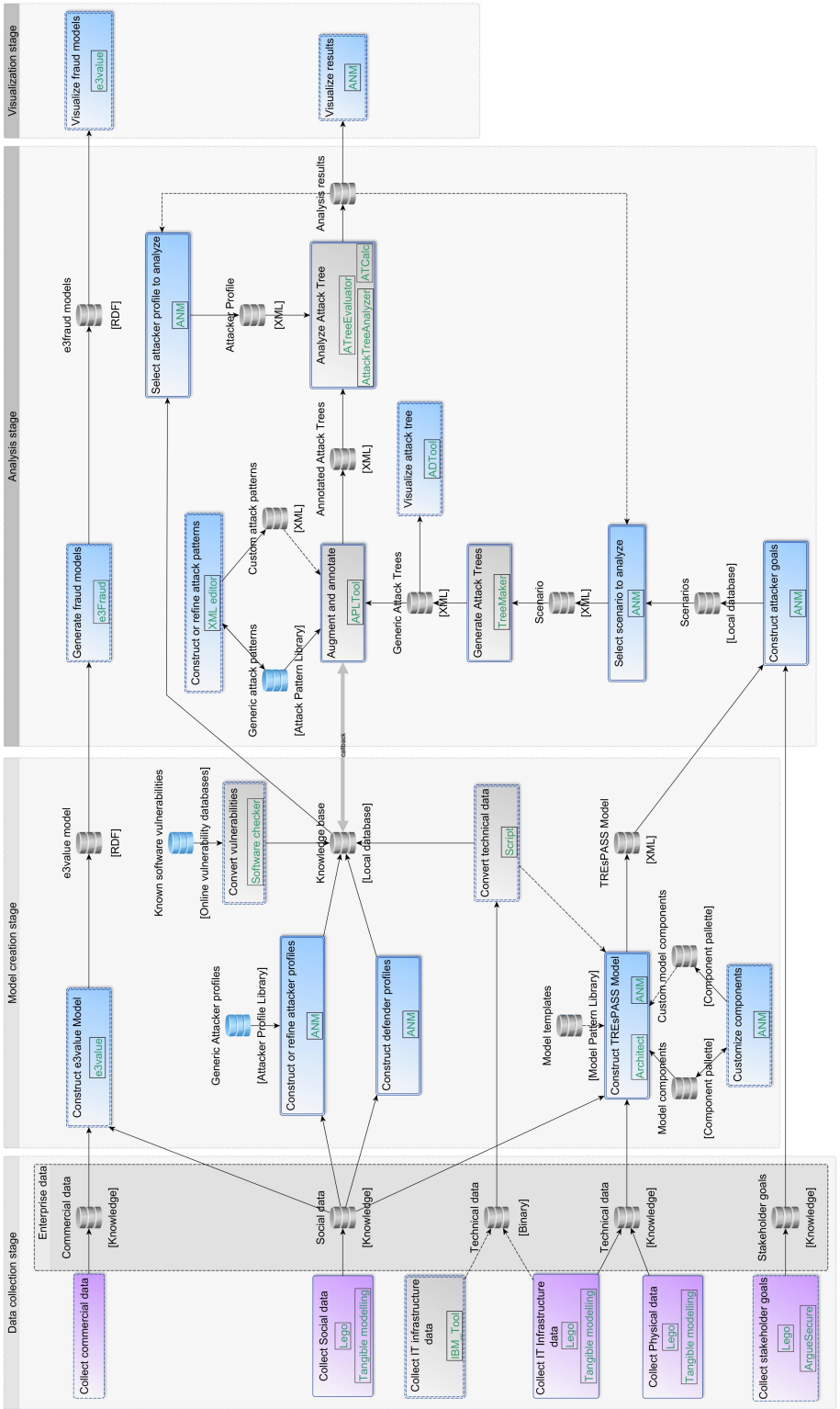

Figure A.2.: Integration diagram for the TRESPASS project.

B. Overview of existing approaches for MPL

B.1. Unified Modeling Language (UML), UMLsec and CORAS

The Unified Modeling Language (UML) is a general-purpose modelling language in the field of software engineering, which is designed to provide a standard way to visualise the design of a system http://en.wikipedia.org/wiki/Unified_Modeling_Language. Since 1997, it has been managed as a standard by the Object Management Group (OMG). Although primarily developed for software design, UML can also be (and has been) applied to model the structure and behaviour of other types of systems. A drawback of UML is the large number of diagram types and the level of detail, which makes UML models less suitable as a tool for communication with non-experts.

Although UML does not explicitly address risk and security aspects, there have been several proposals for adding these aspects to the language. The most prominent one is UMLsec, a lightweight extension to UML for integrating security-related information in UML specifications. This information can be used for model-based security engineering. Most security information is added using stereotypes, which cover many security properties including secure information flow, confidentiality and access control. Using an attacker model these properties can be checked on a model level. UMLsec was first proposed by Jürjens et al. in 2002] and later revised and extended by the same author (Jurjens, 2005). CORAS (Lund, Solhaug, & Stølen, 2011) is the result of an earlier European project, and provides an approach for model-based security analysis based on UML diagrams.

Name	Unified Modelling Language (UML), and the UMLsec and CORAS extensions
Organisation	Object Management Group (OMG)
Link	http://www.uml.org/; http://en.wikipedia.org/wiki/UMLsec
Contents/Scope	Graphical language for IT/software design (structure and behaviour), system design; normative concepts and language; exchange format defined (XMI), machine usable via this format; many different types of diagrams, including workflow and process centric diagrams; UMLsec profile to extend the language with security-specific concepts

Strengths	Graphical representation, expressiveness; large usage in the IT community; common definition of terms; availability of many supporting tools; exchange format defined
Weaknesses	Complex, focus on IT components/software/processes
License	Open (also ISO release)
Format	XML/XMI
Potential use	As UML is widely used in the community, for infrastructure where UML descriptsions are available, the data can be easily collected. Extensions for security modelling such as UMLsec and CORAS improve the link with the TRE$_S$PASS model

B.2. Systems Modelling Language (SysML)

The Systems Modeling Language (SysML) is a general-purpose modelling language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. SysML was originally developed by an open source specification project, and includes an open source license for distribution and use [ref]. SysML is defined as an extension of a subset of the Unified Modeling Language (UML) standard, using UML's profile mechanism http://en.wikipedia.org/wiki/Systems_Modeling_Language.

The advantage of SysML compared to UML is its broader scope, i.e., it is less software-centric, which makes it more suitable for the types of models that are needed in TRE$_S$PASS. Most of the structural and behavioural aspects that can be modelled with the TRE$_S$PASS model have a counterpart in SysML. In addition to this, SysML defines an additional diagram type, the requirement diagram, which may also be useful to model security requirements. SysML does not provide dedicated constructs for modelling risk and security aspects.

Name	Systems Modelling Language (SysML)
Organisation	Object Management Group (OMG); Open Source Specification Project
Link	http://www.omgsysml.org; http://www.sysml.org
Contents/Scope	SysML is based on UML version 2, but extends it with SysML specific diagrams; created to specifically fulfill the requirements for Systems Engineering (see http://www.omgsysml.org/#What-Is_SysML). This includes capturing requirements of components in order to allow analysis and evaluations of systems, as well as communicating information about the system
Strengths	similar to UML; added support for analysis

Weaknesses	not as widely used as UML
License	Open source
Format	XML/XMI
Potential use	Similar to UML. Further evaluation is needed to decide whether this could be of help for system analysis.

B.3. Business Process Model and Notation (BPMN)

For certain security scenarios, it is also relevant to model the workflows that are involved. There is a large variety of languages that can be used for this purpose (including a number of diagram types included in UML). However, the main standard to express business processes and workflows is the Object Management Group's Business Process Model and Notation (BPMN), which provides businesses with a graphical representation that helps in the understanding of a business's internal business procedures, and will give organisations the ability to communicate these procedures in a standard manner. The language can also be used as a front-end for workflow automation by means of a process engine.

BPMN does not have native support for risk and security modelling, although some proposals are available for including these aspects.

Name	Business Process Model and Notation
Organisation	Object Management Group (OMG);
Link	http://www.bpmn.org
Contents/Scope	Capturing of processes, including automated and human processes; BPMN 2 introduced a formal description how elements are executed, therefore aims also to automatically run and track processes in a process server.
Strengths	In wide use in the process community; graphical notation easily understandable by humans, but also formalised for machine usage; aims to connect processes and organisation
Weaknesses	Focus on processes
License	Open definition
Format	XML-based format
Potential use	Enterprises more and more have formally defined processes. Understanding BPMN could allow automatic collection and integration of business processes into the TRE$_S$PASS model

B.4. ArchiMate

ArchiMate, a standard maintained by The Open Group (The Open Group, 2013), is an open and independent modelling language for enterprise architecture that is supported by different tool vendors and consulting firms. The core language provides concepts to model the high structure and behaviour of the business, applications and (information) technology of an organisation, as well as the relationships between those aspects. Moreover, two extensions to the language have been defined, one for modelling the business motivation for the architecture, and one for modelling the implementation and migration aspects. The motivation extension is very suitable to express the stakeholders and their goals, as well as the risk and security principles and requirements.

We have demonstrated that the infrastructure as defined with the core language can be mapped to elements of the TRE$_S$PASS model. Figure 3.6 shows an example.

A white paper of The Open Group (Band et al., 2015) describes how risk and security aspects can be expressed with the ArchiMate language. This paper also proposes specialisations of ArchiMate concepts, mainly from the motivation extension, to model the outcome of a risk assessment, and the requirements for security controls.

Name	ArchiMate
Organisation	The Open Group
Link	http://www.opengroup.org/archimate
Contents/Scope	Enterprise architecture (business, application and technology), business motivation, implementation and migration planning
Strengths	Holistic, integrated view on organisations; user-friendly; large user base; availability of tool support
Weaknesses	Little detail, lack of pre-defined attributes
License	Open standard, commercial licence required for commercial use (tools, courses, consultancy)
Format	Standard XML-based exchange format of The Open Group (https://www2.opengroup.org/ogsys/catalog/C154)
Potential use	Many medium to large information-intensive organisations have ArchiMate models of their enterprise architecture in place, which often includes a description of their (technical) infrastructure. In those cases, relevant data can easily be collected. Also, risk assessment results that have been documented in an ArchiMate model (motivation extension) may be used as input for attacker models.