predict
prioritise
prevent

TRESPASS

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

Deliverable D1.2.2

Final Policy-Specification Language

Project:	TRE$_S$PASS
Project Number:	ICT-318003
Deliverable:	D1.2.2
Title:	Final Policy-Specification Language
Version:	1.0
Confidentiality:	Public
Editor:	René Rydhof Hansen
Cont. Authors:	R.R. Hansen, C.W. Probst
Date:	2015-10-30

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

Members of the TRESPASS Consortium

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

Document History

Authors		
Partner	Name	Chapters
AAU	René Rydhof Hansen	ALL
DTU	Christian W. Probst	ALL

Quality assurance		
Role	Name	Date
Editor	René Rydhof Hansen	2015-10-30
Reviewer	Aleksandr Lenin	2015-10-15
Reviewer	Steve Muller	2015-10-15
Task leader	Lorena Montoya	2015-10-30
WP leader	Christian W. Probst	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

Management Summary

Key takeaways:

- The TRESPASS policy-specification language has been finalised as a combination of policies and processes.
- The TRESPASS policy-specification language can capture local access-control policies and global organisational policies.
- The resulting policy language is designed to be easily extended; we present how to add Datalog and PEAL components to the TRESPASS policy language.

Different policy specification languages offer different possibilities for expressing policies. To choose the right language is impossible, which also is the reason for the existence of so many different languages in the first place. We choose an approach similiar to that chosen in .NET, where programming languages with paradigms as fundamentally different as functional, imperative, and logical, are translated to the same intermediate language.

In the TRESPASS project, the policy-specification language is needed both to model actual security policies, but also as an abstract way to capture properties from the modelled system. The policies defined in a TRESPASS model are used to drive analysis and, in particular, to define the goals that are needed for automated attack generation, a fundamental building block in the assessment of the risk faced by organisations: Intuitively, automated attack generation works by exploring if it is possible for an attacker to reach a state in the system model, where one of the stated security goals can be broken. Technically, this can be implemented by starting from a negated security goal, i.e., a security goal that has been broken by assumption, and then systematically explore the potential ways such a negated security goal could have been reached

In this process, policies play a crucial role since they define which actions are allowed or forbidden, and which credentials are required in order to perform an action.

The goal of this document is to describe the final version of the TRESPASS policy-specification language and to show how it can be extended with features and properties of other policy-specification languages, with special focus on the languages discussed in The TRESPASS Project, D1.2.1 (2014).

1. Introduction

This deliverable is the final report on work performed in Task T1.2 (“Policies in Socio-Technical Security Models”), whose primary goal is to develop mechanisms for specifying policies and relate them to socio-technical security models and to investigate current mechanisms to specify measures such as probability, effort, and impact in policies. Appendix A provides the context for this deliverable in the TRE$_S$PASS project. It describes the overall summary of the project and the TRE$_S$PASS workflow. Figure 1.1 shows the section of the TRESPASS workflow that is adressed by this deliverable.

Different policy specification languages offer different possibilities for expressing policies. To choose the right language is impossible, which also is the reason for the existence of so many different languages in the first place.

Even more importantly, a one-language-only approach, integrating all aspects of all possible languages, is not useful in practice. Our approach has therefore been similiar to that chosen in .NET, where programming languages with paradigms as fundamentally different as functional, imperative, and logical, are translated to the same intermediate language.

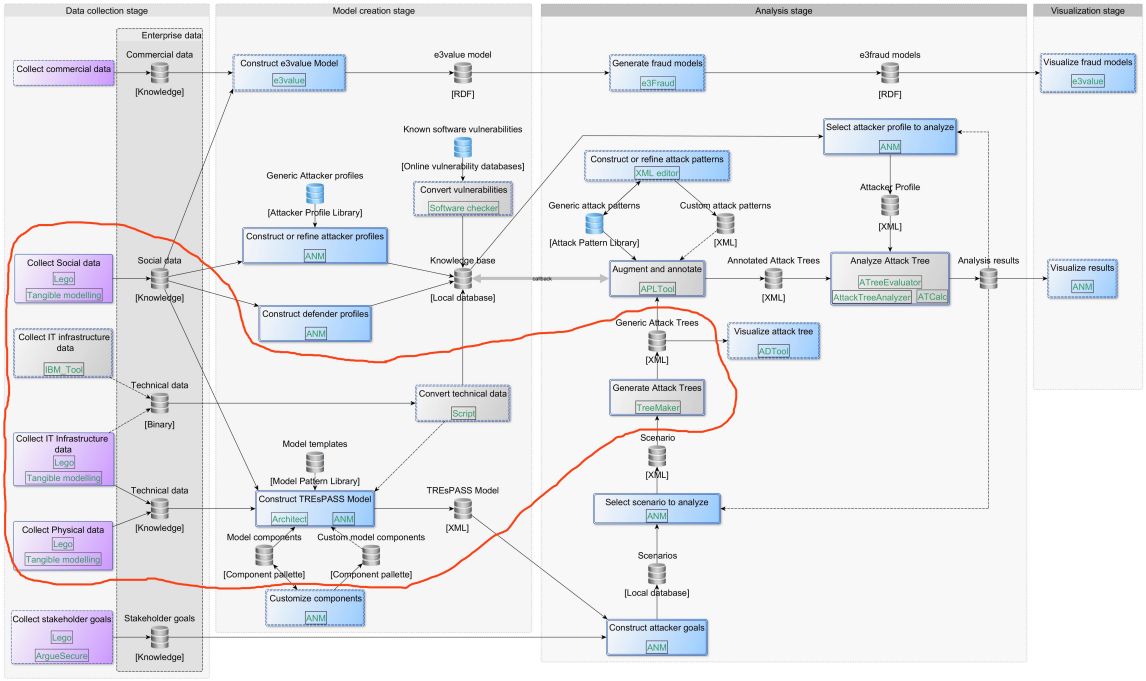

Figure 1.1.: Integration diagram for the TRE$_S$PASS project and the area that this deliverable addresses. Figure A.2 shows a larger version of the diagram.

This deliverable presents this approach and discusses its feasibility and how existing policy specification languages can be mapped to a combination of policies and processes.

We believe that the combination of policies and processes offered by the TRESPASS model empowers users of the TRESPASS tools to choose the right policy language, and to express the necessary policies.

In the following we assume familiarity with the TRESPASS modelling language, as defined in The TRESPASS Project, D1.3.1 (2013), as well as the previous iteration of the TRESPASS policy-specification language, as defined in The TRESPASS Project, D1.2.1 (2014).

1.1. Goals

The main goal of the present deliverable is to formally introduce the TRESPASS policy specification language, based on the current state of the art of mechanisms and languages for policy specification reviewed in a previous deliverable (The TRESPASS Project, D1.2.1, 2014). The semantics and features of this language have a major impact on the precision, expressivity, and applicability of the overall modelling language and the concomitant analyses. A further goal of this deliverable is to illustrate how the important features and properties of the policy languages surveyed in (The TRESPASS Project, D1.2.1, 2014) can be encoded or integrated into the TRESPASS policy-specification language.

1.2. Foreground and Background

While some background material is referenced, notably on the Datalog and PEAL languages, all of the material developed in this deliverable is foreground.

1.3. Document Structure

The rest of this deliverable is structured as follows. After a brief summary and recap of the previous deliverable on policy languages (The TRESPASS Project, D1.2.1, 2014), i.e., what is a security policy and its role in the TRESPASS project, we present the TRESPASS policy-specification language in Chapter 3. After a demonstration of applications of the language to a case studies, Chapter 5 concludes the deliverable.

2. Security Policies

In this chapter we give an overview of the broader context of *security policies*, including their definition and use, before going into the details of defining security policies in the specific context of the TRESPASS project in later chapters. Most of the material in this chapter is adapted from (The TRESPASS Project, D1.2.1, 2014) where a more detailed discussion of these topics can also be found.

2.1. Context and Definition

According to *The American Heritage Dictionary* (1982), a *policy* is a plan or a course of action designed to influence and determine decisions, actions, and other matters. Accordingly, the primary goal of a *security policy* is to maintain the security of critical information resources, often stated in terms of maintaining confidentiality, integrity, and availability of such information resources (Bishop, 2002). Here *confidentiality* is concerned with ensuring that information is only disclosed to those users (subjects) that are authorised to access the information and has a legitimate *need to know*. Conversely, *integrity* is concerned with assurance of the accuracy and consistency of the information. This typically requires strict control of which users, or more generally which *actors*, are allowed to modify a controlled information resource and how they are allowed to modify it. Finally, *availability* is concerned with providing access for authorised users to needed information resources without undue delay.

Having specified a security policy, the next step is to operationalise it. This is most often done using a combination of security mechanisms, typically including authentication, access control, and auditing. Here authentication is the combination of identification of a user and subsequent verification that the user is authorised to perform the desired actions on the controlled information resource. Access control is intended to restrict the actions of authenticated users to ensure that access to a resource can not be misused or leveraged into further, unauthorised, access. Finally, auditing of logs and records made primarily by the implemented security mechanisms facilitates after-the-fact analysis of security breaches and may be used to establish what entities are responsible for a breach. We refer to The TRESPASS Project, D1.2.1 (2014) for a more detailed discussion of access control policies.

As an example, corporate security policies involve the set of laws, rules, and practices that regulate how assets including sensitive information are managed, protected, and distributed within an organisation (HMSO, 1991). To operationalise such data protection, an organisation defines security policies stating how sensitive information will be dealt with.

In the TRESPASS approach, security policies are important not only for modelling purposes, but also for analysis purposes: security policies can be used as a starting point for automated attack generation by exploring ways to break a given policy.

In order to gain a better perspective on security policies, we briefly summarise the framework proposed by Sterne (1991) and previously argued to be a good fit for the philosphy and goals of the TRESPASS approach (The TRESPASS Project, D1.2.1, 2014). According to Sterne (1991) a security policy fits into one of the following three categories:

- *security policy objective*, which may be considered an overarching goal or a "mission statement" for information security. A security policy objective defines, at a high-level, what information resources to protect and what to protect them against; it does not prescribe specific protection mechanisms or describe technical details of attacks. An example of a security policy objective might be 'maximum network availability should be maintained at all times'. While these statements are important, they are very high level and provide limited opportunities for further analysis in the TRESPASS context.
- *organisational security policy*, which delve into more details with security rules, mechanisms, and practices in order to support the security policy objectives set out by an organisation. The organisational security policies are also a natural place to import, implement, and codify all the relevant legislation and compliance measures related to managing and protecting an organisation's information resources. This requires defining criteria for authorising individual users, user roles, conditions for delegation of authority, etc. An organisational security policy is meaningful as long as it provides individuals reasonable ability to determine whether their actions violate or comply with the policy. Continuing the example above, an organisation might declare that only senior staff members should be given a key to the company premises. This would be supporting the security policy objective of maintaining maximum network availability at all times because it limits the number of people potentially having physical access to the building in which such network equipment is located.
- *automated security policy*, which operate at the (low) technical level and involves technical measures and mechanisms employed in order to implement and support organisational security policies. Examples include the access rules defined in a network gateway of firewall to enforce network separation and control.

In a later chapter we discuss how the above categories can be modelled/integrated in the TRESPASS policy-specification language. We refer to The TRESPASS Project, D1.2.1 (2014) for a discussion of the state-of-the-art of policy specification languages as well as a more detailed discussion of their different features and properties.

2.2. The Role of Security Policies within TRESPASS

As has already been mentioned, security policies play a particularly important role within TRESPASS. They are needed not only in order to make models more precise and/or cor-

rect, but also to drive several analyses and the automated attack generation. Intuitively, automated attack generation works by exploring if it is possible for an attacker to reach a state in the system model, where one of the stated security goals can be broken. Technically, this can be implemented by starting from a *negated* security goal, i.e., a security goal that has been broken by assumption, and then systematically explore the potential ways such a negated security goal could have been reached (Kammüller & Probst, 2013; Ivanova, Hansen, & Kammüller, 2015a, 2015b). In particular, this explorative process will eventually show if an attacker could have performed a sequence of actions leading to the broken security goal, i.e., if there is a possible attack.

In previous work, we investigated the extent to which existing approaches to security policy specification overlap with the one proposed within TRESPASS. It was found that the targeted version of the TRESPASS policy specification language was already able to cover all the most influential access control paradigms, notably discretionary, mandatory, and role-based access control (The TRESPASS Project, D1.2.1, 2014). In the present deliverable, we continue and extend that investigation to cover the most important properties of the policy specification languages previously surveyed.

3. The TRESPASS Policy-Specification Language

The TRESPASS modelling language, as described in The TRESPASS Project, D1.1.2 (2015), supports two general mechanisms for modelling and enforcing general security policies, namely *local policies*, concerning the actions of discrete entities such as singel actors, and *global policies*, concerning the overall system state. Both these mechanisms operate at the level of the modelling language and are used to restrict the actions that can be performed *in the model*. Therefore, these mechanisms do not necessarily correspond directly to security policies in the system that is being modelled. Instead, all types of security policies in the system being modelled will be mapped to a mix of local and global policies. By combining local and global policies it is possible to impose fine-grained controls on the reachable states of the model as well as restrict the possible paths between arbitrary states in the model. In the remainder of this chapter, we formally define local and global policies and show how they can be used to express some of the security policies studied in the TRESPASS case studies, but also how they can implement important general features from other well-known policy description languages or even adapt and incorporate other policy description languages. This is important for a number of reasons: instead of trying to capture all possible use cases in one language, which will likely fail, we design our language to be able to cope with evolving requirements, specialised needs, and novel developments. In addition to modelling, both local and global policies form the basis for attack generation by *policy negation* (Kammüller & Probst, 2013), in which the negation of a security policy is used directly to drive the automatic generation of attacker actions needed to reach a state where the negated security policy holds, i.e., a state in which an attack has been successfully performed.

From a high-level perspective, local policies impose fine-grained controls on the particular actions that can be performed by entities in the model, including (models of) actors. Local policies further define what *credentials* are necessary to perform those actions and restrict the set of locations that can be reached by an actor. Conversely, global policies define what states in the model are considered acceptable or secure as well as what states are considered (potentially) insecure. This combination of mechanisms allows us to model, specify, and analyse both functional and state-based properties of the modelled security policies and thereby express most features found in existing policy-specification languages (The TRESPASS Project, D1.2.1, 2014). Furthermore, as we will describe in more detail later in this chapter, the credentials that are required by local and global policies have a very flexible structure and representation, as a term algebra over an appropriate signature, that allows us to directly incorporate elements from other policy-description languages such as PEAL (Crampton, Huth, & Morisset, 2013). This approach makes it

easy both to extend the TRE$_S$PASS policy-specification language with novel policy types, but also to adapt the language for specialised applications, while keeping the core policy description language small enough to be manageable. This point is very well illustrated in the case of PEAL with its focus on (numerical) evidence aggregation, which is a powerful and very expressive paradigm, but it does entail a lot of extra theory and formalisms that are not applicable in all situations. As an aside, with the above division of responsibilities between local and global policies in mind, it is worth noting that local policies then correspond to automated security policies (for the modelling language itself) in Sterne's policy framework (Sterne, 1991), while global policies can be both security policy objectives and organisational security policies.

The remainder of this chapter is organised as follows. We first give a formalisation of local and global policies in Section 3.2, followed by a formalisation of credentials, as used in local and global policies, and discuss how different semantics "backends" can be used to evaluate credentials in Section 3.3.

Especially Sections 3.3.1 and 3.3.2 are noteworthy, as they represent extensions of the TRE$_S$PASS modelling approach. The general concept of extensions is discussed in (The TRE$_S$PASS Project, D1.3.2, 2015).

3.1. Language Specification

As mentioned above, the goal of the TRE$_S$PASS policy-specification language is similar to that of .NET, where programming languages with paradigms as fundamentally different as functional, imperative, and logical, are translated to the same intermediate language. This goal has resulted in an approach that is quite powerful in its own right, but is ideal when used as target for other policy languages.

The main mechanisms in the TRE$_S$PASS policy-specification language are policies and processes: policies specify required credentials and enabled actions, and process specify behaviour. By tying policies and processes together, we can model a wide range of policy scenarios as well as policy languages; the expressivity of this approach is in principle only limited by the expressivity of the process language. As discussed in (The TRE$_S$PASS Project, D1.3.2, 2015), the modelling language used in TRE$_S$PASS can easily be extended, so it can be adapted to requirements of the model.

A formal definition of policies follows in the next section, here it suffices to note that local policies consist of a set of credentials and a set of actions that is enabled, if a user presents the necessary credentials, for example, the policy key : `move` requires a key to move to the location annotated with this policy.

The enabling factor for tying policies and processes together, is the `out` action: it enables actors with the right credentials to output a token in the tuple space of the location protected by this policy. At the same location, one or more processes can wait for "their" token by executing an `in` action, which blocks until the expected input becomes available. For example, consider the policy and processes for different users in a cloud infrastructure

regarding moving a virtual machine or logging in. The policies could enable actors with the root password to output one token, and actors with the user password to output another token:

$$\begin{array}{lcl} password\_root & : & \texttt{out}(\texttt{"move"}) \\ password\_user & : & \texttt{out}(\texttt{"login"}) \end{array}$$

At the same location, two processes would wait for these tokens:

$$\begin{array}{l} \texttt{in}(\texttt{"move"}).P_{movevirtualmachine} \\ \texttt{in}(\texttt{"login"}).P_{login} \end{array}$$

It should be noted that this approach also can model nondeterministic systems by adding two or more processes that all wait for the same token.

3.1.1. Syntax Specification

Figure 3.1 shows the syntax of the TRE$_S$PASS policy specification language for local and global policies. Local policies consist of pairs of credentials and actions, with the semantics that an actor needs to provide all the credentials in order to perform an action. If a policy consists of several pairs of credentials and actions, these are alternative policies. Global policy specify an actor or a variable, which constrain whom the policy should apply to, a set of credentials, and a set of actions, that actors are supposed *not* to perform.

Credentials can be a set of items, data, identity, location, or predicate. An item has an id and can contain other items or data, and data has an id and must either have a value,

$$\begin{array}{rcl}
\textit{LocalPolicies} & := & (\ \textit{credentials}\ :\ \textit{actions}\ )\star \\
\textit{GlobalPolicies} & := & (\ (\ \boldsymbol{actorID} \mid \boldsymbol{variable}\ )\ ,\ \textit{credentials}\ ,\ \textit{actions}\ )\star \\
\textit{credentials} & := & \emptyset \mid \{\ (\ \textit{item} \mid \textit{data} \mid \textit{location} \mid \boldsymbol{actorID} \mid \boldsymbol{pred}\ (\ \textit{argument}\star\ )\ )\star\ \} \\
\textit{item} & := & \boldsymbol{id}\ (\ (\ (\ \textit{item} \mid \textit{data}\ )\ )\ )? \\
\textit{data} & := & \boldsymbol{id}\ (\ \boldsymbol{value} \mid \boldsymbol{variable}\ ) \\
\textit{actions} & := & \emptyset \mid \{\ (\ \textit{in} \mid \textit{out} \mid \textit{move} \mid \textit{eval}\ )\star\ \} \\
\textit{in} & := & \boldsymbol{in}\ (\ (\ \textit{argument}*\ )\ )? \\
\textit{out} & := & \boldsymbol{out}\ (\ (\ \textit{argument}*\ )\ )? \\
\textit{eval} & := & \boldsymbol{eval}\ (\ \boldsymbol{P}\ (\ \textit{argument}*\ )\ )? \\
\textit{move} & := & \boldsymbol{move} \\
\textit{argument} & := & \boldsymbol{value} \mid \boldsymbol{variable} \mid _
\end{array}$$

Figure 3.1.: Syntax of the TRE$_S$PASS policy language.

or be bound to a variable. An identity is matched against the actor's identity, a location specifies where the actor must be located to perform an action, and predicates support specification of properties of actors. Variables provide a further means to tie policies and processes together, as well as credentials. It should be noted that variables are introduced for convenience; for finite models, one can just instantiate all variables with all possible values in policies. In the next section we will see examples for policies such as $\{(\{card(pin(X)), pin(X)\}, \{\mathbf{in}\})\}$, which requires an actor to present a card that stores a pin, and a matching pin, in order to perform the in action.

The enabled actions can also take arguments, that either are values as in the login example above, or variables that are bound to values from the credentials that have enabled the action. The eval action executes a process **P** with some arguments, and the move action permits the actor to move to the location.

3.2. Formalising Local and Global Policies

In this section we formalise the main policy specification and enforcement mechanisms in the TRE$_S$PASS policy-specification language: the local and global policies. The underlying principles have not changed since the last iteration discussed in The TRE$_S$PASS Project, D1.2.1 (2014): both types of policies specify the actions (in the underlying process calculus) that are allowed for a given set of credentials in a particular state of the model (The TRE$_S$PASS Project, D1.2.1, 2014). However, it has been recognised that the notion of credentials is of separate interest and a good candidate for integration with other policy-specification languages. Consequently, we will "factor out" credentials from the description of local and global policies and treat them independently in a later section, where we also examine the flexibility of this approach by incorporating key elements from both the Datalog language (Ceri, Gottlob, & Tanca, 1989) and the PEAL specification language (Crampton et al., 2013).

3.2.1. Local Policies

In the TRE$_S$PASS model, a local policy comprises two parts: the set of *enabled actions* and the required credentials as introduced above. Formally:

$$\begin{aligned} LocalPolicies &= \mathcal{P}\left(RequiredCredentials \times Actions\right) \\ Actions &= \mathcal{P}\left(\{\mathbf{in}, \mathbf{out}, \mathbf{move}, \mathbf{eval}\}\right) \end{aligned}$$

Note that we ignore arguments to the enabled actions from now on, since they are not related to the checking of policies. Intuitively, the credentials describe preconditions that have to be met before the enabled actions may be performed. Typical uses of credentials include

- restricting certain actions to specific locations, e.g., ensuring that confidential data never leaves a secure location;

- restricting actions to a particular actor or role, e.g., allowing only authorised system administrators to upgrade a system; and
- requiring a user to be in possession of a specific (physical or virtual) item, e.g., allowing access only to those with a proper PIN and/or PIN card.

The enabled actions all correspond to specific semantic actions in the process calculus, a variant of Klaim, underlying the semantics of the TRE$_S$PASS modelling language (The TRE$_S$PASS Project, D1.3.1, 2013). The enabled actions seen above correspond to data input or output, moving of actors or processes, and process execution. It should be noted that these actions are the *only* actions that an actor can perform (in the model) and more complex actions (in the modelled system) must be modelled using these basic actions. All local policies are attached to *locations*, which is again a concept intrinsic to the underlying process calculus. Locations (in the process calculus) can be used to model both physical and virtual locations (in the system being modelled) but also any other entities with a specific identity that need to be referenced, e.g., certain physical or digital items or even more abstract concepts such as information known or (physical) items possessed by an actor.

It now remains to define how local policies are enforced. In particular we must specify how a local policy can be used to verify that a given set of actions may be performed by an actor. Intuitively, an actor must provide adequate credentials (with respect to the local policy) to be allowed to perform the concomitant actions, as defined in the local policy. For generality, and in anticipation of later sections, we distinguish between the domain of required credentials and that of provided credentials, denoted $ProvidedCredentials$. The domain of required credentials may be thought of as general access rules, possibly parameterised, and the set of provided credentials as concrete instances (and possibly parameters) of these rules. Since we have yet to formally define the notion of credentials, both required and provided, we shall assume the existence of an acceptance function that can determine if a given set of provided credentials $\chi \in ProvidedCredentials$ is sufficient to satisfy a given set of required credentials; this is denoted $acceptCredential(\chi, \chi')$. Finally, we can define what it means for a set of actions $\alpha \in Actions$ and concomitant provided credentials $\chi \in ProvidedCredentials$ to be *enabled* in a local policy $\Lambda \in LocalPolicies$, denoted $\Lambda \vdash_L (\chi, \alpha)$:

$$\Lambda \vdash_L (\chi, \alpha) \quad \text{iff} \quad \exists (\chi', \alpha') \in \Lambda \colon acceptCredential(\chi, \chi') \ \wedge \ \alpha \subseteq \alpha'$$

For intuition $\Lambda \vdash_L (\chi, \alpha)$ means that policy Λ grants access to exercise any and all actions in α if credentials χ are provided.

Example 3.1 *A local policy formalising that an identity card with a PIN code must be presented along with the corresponding PIN code in order to gain access to a protected location, may be formulated in the following way:*

$$\{(\{card(pin(X)), pin(X)\}, \{\mathbf{in}\})\}$$

The set of credentials above, $\{card(pin(X)), pin(X)\}$*, is a formalisation of the requirement that a user should present a valid access card (denoted* $card()$*) containing a PIN code*

(denoted $pin(X)$, the X is a variable indicating that the specific PIN code is irrelevant, as long as there is one) and should also enter the corresponding PIN code (denoted $pin(X)$, here the X is used to link back to the PIN code found on the access card). The formal details of credentials will be discussed in detail in Section 3.3.

3.2.2. Global Policies

As described in the previous section, local policies are always associated with a particular location that also delimits the scope of a local policy. In contrast, *global policies* specify system-wide policies that must be enforced everywhere, at any time, in the system, i.e., they must hold in all states of the model. As already noted, these correspond more closely to the security policy objectives or the organisational security policies of Sterne than does local policies (Sterne, 1991). Another difference between local and global policies, is that global policies are, by default, *negative* policies in the sense that they specify behaviour that is *not* admissible and states that must *not* be reached in the system. This does not change the expressive power of global policies, it merely makes it more convenient to write typical (global) policies.

A global policy is composed of three parts: the actor affected by the policy, the conditions that must hold in order for the policy to be enforced (formulated as a set of credentials; these will be formalised in a later section), and the set of actions the actor is not allowed to perform:

$$GlobalPolicies = (Actors \cup Vars) \times RequiredCredentials \times Actions$$

where $Actors$ is the set of actors in the model and $Vars$ is a set of variables that can be used and further constrained in the credentials part of the policy. We shall not go into further details with the $Vars$ domain here, but refer to Section 3.3 for details. Thus, a global policy $(a, \chi, \alpha) \in GlobalPolicies$ specifies that in any model state where the conditions (formulated in terms of required credentials) χ hold, actor a is *not* allowed to perform any of the actions in α.

With this, we can now formalise what it means for a global policy $\Gamma \in GlobalPolicies$ to *deny* a given actor $a \in Actors$, providing credentials $\chi \in ProvidedCredentials$, to perform any action from a set of actions $\alpha \in Actions$. We denote this $\Gamma \not\vdash_G (a, \chi, \alpha)$, emphasising the negative nature of global policies:

$$\begin{aligned}\Gamma \not\vdash_G (a, \chi, \alpha) \quad \text{iff} \quad & (\exists (a', \chi', \alpha') \in \Gamma \colon a = a' \ \wedge \ acceptCredential(\chi, \chi') \ \wedge \ \alpha \cap \alpha' \neq \emptyset) \vee \\ & (\exists (X, \chi', \alpha') \in \Gamma \colon acceptCredential(\chi, [X \mapsto a]\chi') \ \wedge \ \alpha \cap \alpha' \neq \emptyset)\end{aligned}$$

where $[X \mapsto a] \in Subst$ is a *substitution* of the variable $X \in Vars$ for the (concrete) actor $a \in Actors$.

Example 3.2 *A global policy forbidding all employees access to a certain location can be written as*

$$\{(X, \{isEmployee(X)\}, \{\mathbf{in}\})\}$$

stating that an actor, in fact any *actor as represented by the variable X, is not allowed to perform an* **in** *action, if the predicate $isEmployee$ is true for X, assuming that the predicate $isEmployee$ is a valid credential, as discussed in Section 3.3.*

One of the most common uses of global policies is to restrict the set of locations an asset can legally reach or be in, e.g., confidential information must not be able to reach an "insecure" location or the keys to the server room are not allowed to leave the building. Although such policies can be expressed as global policies by restricting the actions of certain actors, we define the more convenient notion of *global location policies* to capture such policies in a more natural manner:

$$GlobalLocationPolicies = Asset \times Location$$

where $Asset$ and $Location$ is the set of assets and locations in the model respectively. Intuitively, a global location policy $(a, l) \in GlobalLocationPolicies$ specifies that the asset a is never allowed to reach location l.

3.3. Formalising Credentials

Following the approach described in The TRESPASS Project, D1.2.1 (2014), we formalise credentials as sets of terms from the term algebra generated from a model-specific signature (Meinke & Tucker, 1992). This formalisation of (the syntax of) credentials gives a uniform representation that is very flexible and expressive. It furthermore allows for many different semantics to be used for credentials, e.g., to include domain specific features and properties or to facilitate attack generation.

The base signature of the term algebra representing credentials for a given model is model-specific and will contain elements representing aspects specific to the model. In the following we therefore assume the existence of a relevant signature $\Sigma = (S_\Sigma, F_\Sigma)$ where S_Σ is the set of sorts and F_Σ is the set of function symbols over the sorts in S_Σ; we shall not go into more details with defining signatures and term algebras here, but refer instead to the comprehensive treatment by Meinke and Tucker (1992) (also, see below for an example). Thus, we can now give the formal definition of required credentials:

$$RequiredCredentials = \mathcal{P}\left(T\left(\Sigma, Vars\right)\right)$$

where $T(\Sigma, Vars)$ is the term algebra generated by the signature Σ and variables found in $Vars$. Variables are needed to express generic or parameterised policies. Variables are also allowed to stand for an actor in a global policy. Such a variable can then be further restricted in the credentials of the global policy following the above formalisation.

Term algebras are very flexible and easy to use for describing the syntax of credentials, since they do not presuppose any particular (kind of) underlying semantics. Furthermore, it is straightforward, if tedious, to encode and/or convert the syntax the policy-description languages surveyed in The TRESPASS Project, D1.2.1 (2014) into term algebra notation.

Since the above term algebra defines only the syntax of required credentials, we still need to define the semantics. As noted in a previous section, required credentials define a set of conditions that must be met by the set of concrete credentials provided (or presented) by the actor in question. Thus, we next formalise the notion of *provided credentials*. Given the above use of term algebras (with variables), it is natural to use the *ground terms* from that algebra as the set of provided credentials:

$$ProvidedCredentials = \mathcal{P}\left(T\left(\Sigma, \emptyset\right)\right)$$

Thus, the set of provided credentials is the set of required credentials that contain no variables.

Verifying that a set of provided credentials $C \in ProvidedCredentials$ is sufficient to meet the conditions set by the required credentials $R \in RequiredCredentials$ can then be formalised using first order unification of terms, as defined by Robinson (1965): if it is possible to find a unifying substitution for R and C such that the resulting required credentials are all included in the provided credentials, then the provided credentials are sufficient, i.e., if there exists a substitution σ such that $\sigma R \subseteq C$:

$$acceptCredential(C, R) \quad \text{iff} \quad \exists \sigma \in Subst\colon \sigma R \subseteq C$$

where $Subst = Vars \to T(\Sigma, \emptyset)$, the domain of all substitutions from variables to ground terms. In Robinson (1965) it is shown how to algorithmically determine the existence of such a substitution and, if it does, how to compute the most general such substitution. We refer to Meinke and Tucker (1992) for a full text-book treatment of all the formal details concerning signatures, term algebras, and unification.

Example 3.3 *Recall from Example 3.1 a local policy specifying the use of PIN cards and PIN codes:* $\{(\{card(pin(X)), pin(X)\}, \{\mathbf{in}\})\}$*. The simplest possible signature for realising this local policy consists of a single sort* $\star$ *and the function symbols* $pin()$ *and* $card()$ *as well as the natural numbers as constants (represented as 0-ary function symbols):*

$$S_\Sigma = \{\star\} \qquad \textit{and} \qquad F_\Sigma = \{\texttt{card}\colon \star \to \star, \texttt{pin}\colon \star \to \star, n\colon \star\}$$

where n *is a natural number. A more sophisticated signature could have a sort representing PIN codes and a sort representing access cards etc. This approach would allow for finer control over the set of terms generated by the term algebra. However, for many purposes, a singleton sgnature will suffice.*

We can now easily see that the required credentials of the targeted local policy, is indeed a set of terms from the generated term algebra, and thus a valid formulation of the required credentials:

$$R = \{card(pin(X)), pin(X)\}$$

We can now use the unification based resolution, cf. (Robinson, 1965), to determine if the following set of provided credentials are sufficient to gain access:

$$C = \{card(pin(1234)), pin(1234)\}$$

It is easy to see that the substitution $\sigma = [X \mapsto 1234]$ *applied to* R *yields*

$$\sigma R = \{card(pin(1234)), pin(1234)\} \subseteq C$$

and thus access is granted.

This approach to verification of credentials has the benefit that it is based on mature, well-known theory, is easy to understand, and is easy to implement. However, it also lacks structure for some commonly wanted modelling tasks, e.g., checking in a database if an actor is also an employee or if an actor is also a customer. Although the approach described above could be extended to also cover these particular situations, e.g., by adding a *knowledge base* (or database) component that could be checked seperately, we instead show, in the following section, how the Datalog language can be used to generalise our approach.

3.3.1. Extension: Datalog

Datalog is a fully declarative database query language for relational databases, with a formal semantics rooted in the logic programming language paradigm (Ceri et al., 1989). The combination of formal semantics, succinct syntax, and efficient algorithms for analysis and execution of Datalog programs makes it very well-suited for our purpose: specifying credentials.

Intuitively, a Datalog program can be seen as a number of logical *inference rules*, specified as *Horn clauses* (sometimes called the *intensional database*), working over a set of ground *facts*, specified as (ground) Datalog rules (sometimes called the *extensional database*). This fits very well into the TRESPASS model since, as mentioned above, it is often convenient to define a knowledge-base with simple facts and relationships apparent in a model, e.g., that an actor is an employee and that a given employee is also a system administrator. In Datalog it is usual to define the notion of *goals* that can be evaluated relative to a given knowledge-base and concomitant program. Goals are themselves Datalog programs and can be seen as ad-hoc database queries into the combined database created by the program and knowledge-base. In the following, we let $KnowledgeBase$ and $DatalogPrograms$ denote the set of knowledge-bases and programs respectively.

We shall not go into further details concerning the semantics or evaluation of Datalog programs here, merely refer to Ceri et al. (1989) and assume the existence of an evaluation relation: $K, P \models_{\text{Datalog}} Q$ denoting that the goal $Q \in DatalogPrograms$ can be satisfied by the program $P \in DatalogPrograms$ over the knowledge-base $K \in KnowledgeBase$. With these definitions, we can now establish the connection to credentials: we use Datalog programs for the required credentials, providing maximal expressivity and flexibility:

$$RequiredCredentials = DatalogPrograms$$

It is then natural to choose ground Datalog programs for provided credentials:

$$ProvidedCredentials = \{\, p \in DatalogPrograms \mid \mathrm{FV}(p) = \emptyset \,\}$$

Finally, we assume the existence of a model-wide knowledge-base $K \in KnowledgeBase$. While it would be possible to attach knowledge-bases both to required and/or provided credentials, the typical use cases in TRESPASS, e.g., an employee database, correspond better to a "global" knowledge-base.

Defining when a set of provided credentials satisfies a set of required credentials, can then be defined simply in terms of Datalog program (with a goal) evaluation:

$$acceptCredential(C, R) \quad \text{iff} \quad K, R \models_{\text{Datalog}} C$$

where $K \in KnowledgeBase$ is the model-wide knowledge-base dicussed above. It is important to note that, similar to first order unification discussed in the previous section, there are efficient algorithms for evaluating Datalog programs and numerous implementations. We refer to Ceri et al. (1989) for details.

Example 3.4 *We expand on Example 3.2 and define a global policy forbidding employees to use (credit-/pin-)cards belonging to clients:*

$$\{(X, \{isEmployee(X), isClient(Y), card(Y, pin(Z))\}, \{\mathbf{in}\})\}$$

where the facts $isEmployee(\cdot)$, $isClient(\cdot)$, and $card(\cdot, pin(\cdot))$ are defined in the global knowledge-base. This policy is relatively straigthforward and could also, with some work, have been specified in the simpler approach of first order unification of terms. However, the global knowledge-base and the use of full Datalog for required credentials, makes it more convenient and possible to formulate more complex policies, e.g., taking advantage of the fact that Datalog can express transitive closures, it would be possible to specify that family members of employees are also not allowed to use credit cards belonging to clients:

$$\{(X, \{isFamily(X, Y), isEmployee(Y), isClient(Z), card(Z, pin(W))\}, \{\mathbf{in}\})\}$$

where the relation $isFamily(\cdot, \cdot)$ is in the knowledge-base.

In addition to the pure version, numerous extensions of Datalog have been studied in detail, most notably the addition of *negation* (Ceri et al., 1989). It is also possible to provide interpretations of Datalog that allows programs (and thus goals) to include arithmetic and boolean expressions; these are of course very useful for specifying policies that include quantitative measures, e.g., cost or time. In the following section, we discuss a different approach to policies including quantities, namely the PEAL policy-description language (Crampton et al., 2013), and show how PEAL can also be integrated into the TRESPASS policy specification language.

3.3.2. Extension: PEAL

In the PEAL policy-specification language, policy decisions are made on the basis of *scores* that are aggregated or computed whenever certain *evidence* is present (or absent). The language was discussed in more detail in The TRESPASS Project, D1.2.1 (2014) and

we will therefore not go into further details here, except to note that PEAL was designed in such way that it would specifically be possible to embed it into other (policy-specification) languages, such as the TRESPASS policy-specification language.

Using the same approach as for integrating Datalog, it is trivial to embed PEAL. We take the required credentials simply to be the *scoring rules* of the PEAL language:

$$RequiredCredentials = PEALScoringRules$$

Since PEAL rules can contain variables that must be assigned a concrete value in order to evaluate the scoring rules, we take as provided credentials a mapping, or assignment, of PEAL variables to concrete values:

$$ProvidedCredentials = Vars \to Val$$

Now, checking if a set of provided credentials $C \in ProvidedCredentials$ satisfies a set of required credentials $R \in RequiredCredentials$ is reduced to evaluating the corresponding PEAL terms:

$$acceptCredential(C, R) \quad \text{iff} \quad R \models_{\text{PEAL}} C$$

where $C \models_{\text{PEAL}} R$ denotes the PEAL evaluation of scoring rules in the environment C.

As an alternative solution, it would be possible to also have a global environment in which to evaluate a set of required credentials *in addition* to the environment defined by the provided credentials. This would afford some of the convenience of the knowledge-base discussed in the previous section for Datalog-based credentials.

Example 3.5 *To illustrate the use of PEAL credentials, we re-iterate the PEAL example from The TRESPASS Project, D1.2.1 (2014, Section 4.3.1.2) can be reformulated in our approach. The example is atypical, since the specified policy does not specifically allow or prohibit a particular action, but instead defines when an alert should be sent.*

The PEAL scoring rules are as follows:

$$\begin{aligned} b_1 &= \min(\text{if } suddenSpendingStop\ 1) \quad \text{default } 0 \\ b_2 &= +\,((\text{if } charityReferral\ 0.4)\ (\text{if } accountHeld\ 0.4) \\ &\qquad (\text{if } writtenAgreement\ 0.2)\ (\text{if } spokenAgreement\ 0.1)\ (\text{if } familyAgreement\ 0.1)) \\ &\quad \text{default } 0 \\ cond &= 0.9 < \min(b_1, b_2) \end{aligned}$$

The threshold defined by the $cond$ expression requires a value greater than 0.9 for the alert to be raised depending on the minimum of return values from policies b_1 and b_2. Thus, the sum of the condition results of policy b_2 and the result of b_1 must exceed 0.9. This encoding can be summarised as follows: an alert may be raised if an actor that holds an account (as recorded by the $accountHeld$ variable) which was set up through a referral from a charity (the $charityReferral$ variable) suddenly stops spending (the $suddenSpendingStop$ variable) and the actors gave written permission (the $writtenAgreement$ variable) or the actor gave a spoken agreement (the $spokenAgreement$ variable) and *the family agrees (the*

$familyAgreement$ variable). We refer to the original example for context and details (The TRESPASS Project, D1.2.1, 2014).

Assuming a TRESPASS model in which raising an alert is modelled as an $\mathbf{eval}$*-action performed at a location encoding the charity. This location will then have a local policy for raising an alert:*

$$\{(R_{\text{Alert}}, \{\mathbf{eval}\})\}$$

where R_{Alert} denotes the PEAL scoring rules above. In order to concretely evaluate if/when to send an action, an actor must provide credentials that assign values to all the variables discussed above:

$$C = [accountHeld \mapsto \texttt{true}, charityReferral \mapsto \texttt{true}, suddenSpendingStop \mapsto \texttt{true}, \\ writtenAgreement \mapsto \texttt{false}, spokenAgreement \mapsto \texttt{true}, familyAgreement \mapsto \texttt{false}]$$

In this case, the threshold value is computed to be 0.9 *and thus the* $\mathbf{eval}$ *action can* not *be performed and therefore an alert is not raised.*

Interestingly, the above example uses numeric values to encode what is, essentially, a logical problem which can be easily formulated as a Datalog program. However, for more involved policies, it may be impossible to gain an overview of the overall outcome of a policy. In such cases "local" scoring rules that focus on aggregating single pieces of evidence are likely more appropriate.

3.3.3. Policies, Credentials, and Attack Generation

Policies and credentials play a major role in TRESPASS. In the model they enable modellers to express access control to, e.g., buildings, rooms, servers, or data. They represent thus an important tool to document security precautions in an organisation. Policies also play the role of documenting organisational policies, for example, expectations to employees. Such a policy describes a state or actions which are disallowed in the system, and are expected to be enforced system-wide. While in practice there will be many global policies, we assume without loss of generality that only one such policy exists; for several policies our approach generates individual attacks and combines them with disjunction nodes.

Policies of either kind are essential in the TRESPASS project for identifying attacks through policy invalidation (Kammüller & Probst, 2013; Ivanova et al., 2015a, 2015b). Policy invalidation operates on global, organisational policies. On a high level, our approach for invalidating a policy consists of four basic steps:

1. Choose the policy to invalidate, and identify the possible actors who could do so; these are the potential attackers.
2. Identify a set of locations where the prohibited actions can be performed. Since there might be several possible actions, this results in a set of pairs of location and action.

3. Recursively generate attacks for performing these actions. This will also identify required assets to perform any of these actions, and obtain them.
4. Finally, move to the location identified in the second step and perform the action.

Step 3 requires typically that credentials are obtained as the result of policies that need to be fulfilled for performing an action. For attack generation, one of the fundamental enabling functionalities in TRE$_S$PASS, policies and credentials therefore are of uttermost importance.

4. Case Study Policies

In this chapter we give a short example for some of the policies that occur in the Cloud Infrastructure use cases in the TRESPASS project.

Assessing risk in cloud infrastructures is difficult. Typical cloud infrastructures contain potentially thousands of nodes that are highly interconnected and dynamic. Another important component is the set of human actors who get access to data and computing infrastructure. The cloud infrastructure therefore constitutes a socio-technical system. Attacks on socio-technical systems are still mostly identified through expert brainstorming. However, formal risk assessment for systems including human actors requires modelling human behaviour, which is difficult at best.

Figure 4.1 shows an example for a very simplified cloud scenario, containing of a small organisation comprised of two rooms and several servers, only one of them shown in the figure. We assume that only Terry has access to the datacenter, Sydney has logical access to all elements of the cloud infrastructure, and Ethan is the rightful owner of a secret file *fileX* stored on virtual machine *VM1*.

There are a number of policies in place, for example to control access to the physical rooms or the computers, and also to control the hypervisors of the cloud infrastructure. Every relevant aspect of the underlying socio-technical security model is translated to Datalog during attack generation to identify where credentials are located, which credentials are missing, etc. Translating the model into Datalog is only necessary to support policy resolution and attack identification, which relies on the former.

For example, the fact that Terry has a keycard and a matching pin is modelled by

```
isActor('Terry').
isItem('id001').
hasName('id001','card').
isData('id002').
hasName('id002','pin').
contains('Terry','id001').
contains('Terry','id002').
contains('id001','id002').
```

The first lines specify that *Terry* is an actor, that *id001* is an item with name *card*, and that *id002* is a data item with name *pin*. The last three lines specify that Terry "contains" *id001* and *id002*, meaning that he has the card and knows the pin. Finally, the card also contains the pin; this is required to be able to test the pin.

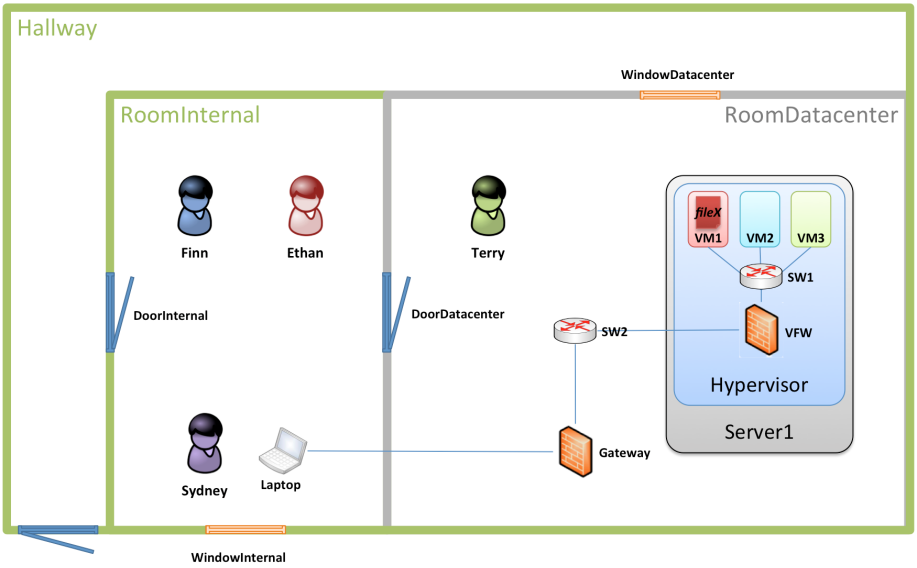

Figure 4.1.: Simplified cloud scenario

If the RoomInternal has a policy that requires only a card to get in, while the server room requires a card and a matching pin, this is translated to Datalog by the attack generation like this:

```
isPolicy('id003').
hasName('cred001','card').
requires('id003','cred001').
enables('id003', 'move').
contains('DoorInternal','id003').
isPolicy('id004').
hasName('cred002','pin').
requires('id004','cred001').
requires('id004','cred002').
contains('DoorDataCenter','id004').
enables('id004', 'move').
```

It should be noted that the TRE$_S$PASS policy language supports the use of variables; this feature is not considered here for clarity since variables can be unfolded due to the finite nature of models.

Also the network part of the cloud scenario contains numerous policies, for example to model a switch. The nodes that are directly connected to the switch can output packets to the switch, and the processes at the switch decide where to send the data traffic. The policy at the switch looks like this:

```
VM1 : {out("IP",,,,,,)}
VM2 : {out("IP",,,,,,) }
VFW : {out("IP",,,,,,) }
```

stating that an actor or process located at location VM1 or VM2 or VFW may output a tuple that begins with the string “IP”. At the switch location there are also a number of processes:

```
in("IP", !srcAddr, 192.167.1.1, !msg).out("IP", srcAddr, 192.167.1.1, msg)@VM1
in("IP", !srcAddr, 192.167.1.2, !msg).out("IP", srcAddr, 192.167.1.2, msg)@VM2
in("IP", !srcAddr, !dstAddr~0.0.0.0/0, !msg).out("IP", srcAddr, dstAddr, msg)@VFW
```

The first process is triggered by tuples that have VM1 as destination, and sends it there, the second process does the same for VM2, and the third process for default routes, that are sent to VFW.

In the Datalog translation, this looks like this for the VM1 policy; the others look similar:

```
isLocation('id005').
hasName('id005','VM1').
isPolicy('id008').
requires('id008','id005').
enables('id008','act001').
hasName('act001','out').
hasArgument('act001',1,'IP');
```

and for the processes the generated Datalog looks similar. For the in action this code is generated:

```
isAction('act002')
hasName('act002','in').
isInput('arg001').
isInput('arg002').
hasName('arg001','SrcAddr').
hasName('arg002','msg').
hasArgument('act002',1,'IP').
hasArgument('act002',2,arg001).
hasArgument('act002',3,'192.167.1.2').
hasArgument('act002',4,arg002).
hasTarget('act002','SW').
```

and for the out action, this code:

```
isAction('act003')
hasName('act003','out').
isOutput('arg001').
isOutput('arg002').
hasName('arg001','SrcAddr').
hasName('arg002','msg').
```

```
hasArgument('act003',1,'IP').
hasArgument('act003',2,arg001).
hasArgument('act003',3,'192.167.1.2').
hasArgument('act003',4,arg002).
hasTarget('act003','VM1').
```

While this translation in isolation looks slightly difficult, it should be noted, that the input is structured code, which significantly eases the automatic generation.

The translation to Datalog simplifies many tasks in the TRE$_S$PASS process. For example, the query where an actor can get a card and a pin looks like this:

$$?-hasName(?var0,'card'), contains(?var1,?var0), hasName(?var2,'pin'), contains(?var3,?var2)$$

This query identifies all tuples of var0 and var2 where the card and the pin are located; this information is latter used in the attack generation to identify the necessary steps in attacks.

5. Conclusions

This deliverable presents the final version of the TRE$_S$PASS Policy Specification Language. The language has been designed to be an expressive middle layer for different kinds of policy languages. Similar to the .NET middle layer, the TRE$_S$PASS Policy Specification Language is rather powerful and is well suited to be the translation goal for other languages.

The expressivity is achieved by mixing policies and processes. Policies may allow certain actions that eventually result in output at a location, and due the separation of concern between policies and processes, the action may trigger one or more processes.

Internally, policies and processes are added to a DataLog-based knowledge base. This internal representation is the basis for representing other policy languages. The important insight is that any policy language that can be represented in DataLog can be used as replacement in the TRE$_S$PASS model. It should be noted that it is not this replacement we want to achieve. As shown for PEAL (Crampton et al., 2013), we can actually use the DataLog goal for translating other language paradigms.

References

The american heritage dictionary [Dictionary]. (1982). Houghton Mifflin,.

Bishop, M. (2002). *Computer security: Art and science* [Book]. Boston USA: Addison-Weley.

Ceri, S., Gottlob, G., & Tanca, L. (1989, March). What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, *1*(1), 146–166. doi: 10.1109/69.43410

Crampton, J., Huth, M., & Morisset, C. (2013, October). *Policy-Based Access Control from Numerical Evidence* (Tech. Rep. No. 2013/6). London, SW7 2AZ, England: Imperial College London, Department of Computing. Retrieved from http://www.doc.ic.ac.uk/research/technicalreports/2013/DTR13-6.pdf (Retrived on 26 October 2015)

HMSO. (1991). *Information technology security evaluation criteria* [Standard]. Department of Trade and Industry.

Ivanova, M. G., Hansen, C. W. P. R. R., & Kammüller, F. (2015a). Attack tree generation by policy invalidation. In R. N. Akram & S. Jajodia (Eds.), *Proceedings of the 9th wistp international conference on information security theory and practice (wistp2015)* (Vol. 9311, p. 249-259).

Ivanova, M. G., Hansen, C. W. P. R. R., & Kammüller, F. (2015b). Transforming graphical system models to graphical attack models. In *Proceedings of the second international work- shop on graphical models for security (gramsec 2015).*

Kammüller, F., & Probst, C. (2013). Invalidating policies using structural information. In *2nd international ieee workshop on research on insider threats, writ'13.* IEEE. (Co-located with IEEE CS Security and Privacy 2013)

Meinke, K., & Tucker, J. V. (1992). Universal algebra. In S. Abramsky, D. Gabbay, & T. Maibaum (Eds.), *Handbook of logic for computer science* (Vol. I: Mathematical Structures, pp. 189–411). Oxford University Press.

Robinson, J. A. (1965, jan). A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, *12*(1), 23–41. doi: 10.1145/321250.321253

Sterne, D. F. (1991). On the buzzword "security policy". In *Research in security and privacy, 1991. proceedings., 1991 ieee computer society symposium on* (pp. 219–230).

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

The TRE$_S$PASS Project, D1.2.1. (2014). *Initial policy-specification language.* (Deliverable D1.2.1)

The TRE$_S$PASS 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)

A. Project Summary

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

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

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

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

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

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

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

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

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

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

A.1. Case Studies

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

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

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

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

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

A.2. Overview of TRESPASS Integration

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

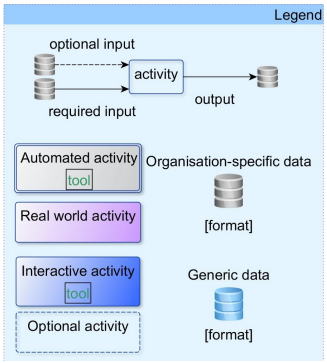

Figure A.1.: Legend for the Integration diagram in Figure A.2.

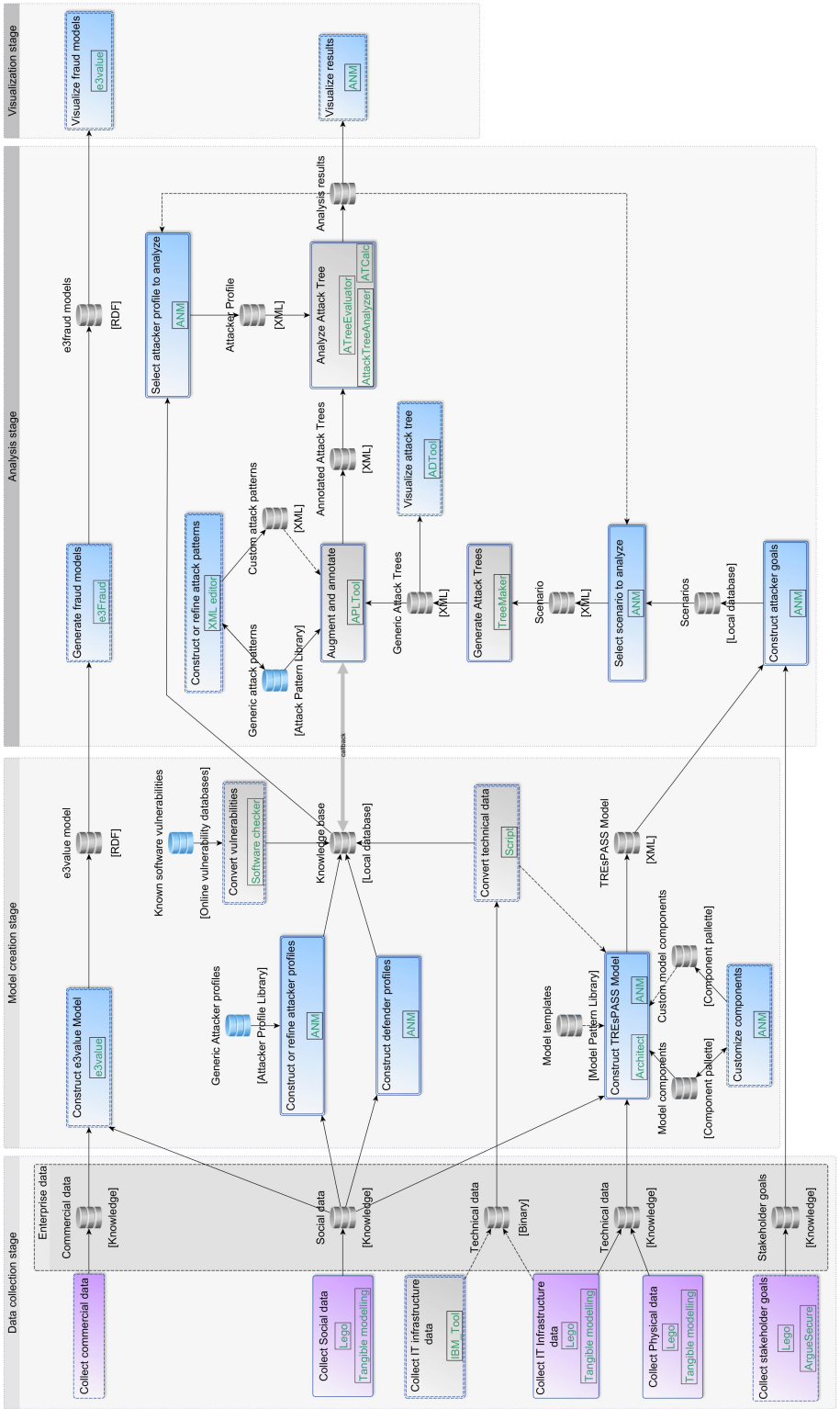

Figure A.2.: Integration diagram for the TRESPASS project.