Problem-based Security Requirements Elicitation and Refinement with PresSuRE[*]

Stephan Faßbender, Maritta Heisel and Rene Meis

University of Duisburg-Essen, paluno - The Ruhr Institute for Software Technology
firstname.lastname@paluno.uni-due.de

Abstract. Different reports on cybercrime, which were published recently, indicate an ever-increasing number of security incidents related to IT systems. Many attacks causing the incidents abuse (in)directly one or more security defects. Fixing the security defect once fielded is costly. To avoid the defects and the subsequent need to fix them, security has to be considered thoroughly when developing software. The earliest phase to do so is the requirements engineering in which security threats should be identify early on and treated by defining sufficient security requirements. In a previous paper [1], we introduce a methodology for Problem-based Security Requirements Elicitation (PresSuRE). PresSuRE provides a computer-aided security threat identification. The identification is based on the functional requirements for a system-to-be. Still, there is a need for guidance on how to derive security requirements once the threats are identified. In this work, we provide such guidance extending PresSuRE and its tool support. We illustrate and validate our approach using a smart grid scenario provided by the industrial partners of the EU project NESSoS.

Key words: security analysis, problem frames, requirements elicitation

1 Introduction

Recently, there has been an increase of reported security incidents hitting large software systems. For example, in the report on cybercrime for the year 2013 published by the federal criminal police office of Germany, the authors state that 64426 security incidents were reported in Germany [2]. This is an increase by 70 percent with respect to 2008[3]. Moreover, particular types of attacks which aim at companies increased much more. For example, data manipulation and computer sabotage incidents in companies increased by 18 percent with respect to 2012 and 578 percent with respect to 2008. These numbers are limited to Germany, but, for example, Norton reports a world wide damage of 113 billion US dollar in 2013 due to security incidents [4]. Hence, the need for secure IT systems is staggering.

[*] Part of this work is funded by the German Research Foundation (DFG) under grant number HE3322/4-2 and the EU project Network of Excellence on Engineering Secure Future Internet Software Services and Systems (NESSoS, ICT-2009.1.4 Trustworthy ICT, Grant No. 256980).

Not all of the security incidents are directly related to security defects in an IT system, but many attacks abuse indirectly or directly one or more security defects. Hence, these security defects need to be fixed. But fixing the security defect causing the incident is costly. Fixing a defect when it is already fielded is reported to be up to eighty times more expensive than fixing the corresponding requirements defects early on [5,6]. Thus, security issues should be detected as early as possible for a system-to-be. Therefore, it is crucial for requirements engineers to identify security threats, and to refine the threats into security requirements. But eliciting good requirements is not an easy task [7], even more with regard to security, as most requirements engineers are not security experts in the first place.

In an previous work of ours, we propose a method called problem-based security requirements elicitation (PresSuRE), which guides a requirements engineer through the process of eliciting a set of security requirements in collaboration with the stakeholders of the system-to-be and security experts [1]. PresSuRE has several benefits. It does not require the requirements engineer to have a security background. It does not require any preliminary security requirements and security relevant information. It lowers the effort by providing tool support for semi-automated modeling and an automated security analysis. And PresSuRE is completely guided by a detailed process.

PresSuRE is based on the same idea of deriving information flows from functional requirements like the problem-based privacy analysis (ProPAn) [8], but changes the analysis to be suitable for security. The analysis and elicitation is based on a complete set of functional requirements for a system-to-be. The method is accompanied with tool-support[1]. PresSuRE is based on the problem frame notation introduced by Jackson [9]. Problem frames are suitable as input for a semi-automated analysis, as they have a predictable structure, underlying semantics, and support focusing on parts of the system-to-be.

But PresSuRE, as reported in the previous work, only gives guidance in detail for steps necessary for analyzing the system-to-be for security threats. But a description how to derive and model initial security requirements, and how to analyze if the security requirements are sufficient regarding the found threats is still missing. Hence, in this paper we provide such guidance by extending PresSuRE.

We briefly describe the case study (Section 2) we use for the running example and the validation. The problem frame notation is explained in Section 3. Section 4 introduces the running example, which is used for the rest of the paper. The PresSuRE method as introduced in [1] is briefly explained in Section 5. In Section 6, we describe the our new extension for deriving security requirements and in Section 7 PresSuRE is validated. In Section 8 related work is discussed, and the final conclusion is drawn in Section 9.

2 Case Study

To illustrate the application of the PresSuRE method, we use the real-life case study of smart grids. As sources for real functional and quality requirements, we consider diverse

[1] http://www.uml4pf.org/ext-pressure/installation.html

documents such as "Application Case Study: Smart Grid" provided by the industrial partners of the EU project NESSoS[2], the "Protection Profile for the Gateway of a Smart Metering System" [10] provided by the German Federal Office for Information Security , and "Requirements of AMI (Advanced Multi-metering Infrastructure") [11] provided by the EU project OPEN meter[3].

To use energy in an optimal way, smart grids make it possible to couple the generation, distribution, storage, and consumption of energy. Smart grids use information and communication technology (ICT), which allows for financial, informational, and electrical transactions.

We define the terms specific to the smart grid domain and our use case in the following. The *smart meter gateway* represents the central communication unit in a *smart metering system*. It is responsible for collecting, processing, storing, and communicating *meter data*. The *meter data* refers to readings measured by smart meters regarding consumption or production of a certain commodity. A *smart meter* represents the device that measures the consumption or production of a certain commodity and sends it to the gateway. An *authorized external entity* can be a human or an IT unit that communicates with the gateway from outside the gateway boundaries through a *wide area network (WAN)*. The *WAN* provides the communication network that interconnects the gateway with the outside world. The *LMN (local metrological network)* provides the communication network between the meter and the gateway. The *HAN (home area network)* provides the communication network between the consumer and the gateway. The term *consumer* refers to end users of commodities (e.g., electricity).

3 Problem-Oriented Requirements Engineering

Jackson [9] introduced the concept of *problem frames*, which is concerned with describing, analyzing, and structuring software development problems. A problem frame represents a class of software development problems. It is described by a *frame diagram*, which consists of domains, interfaces between them, and a requirement. Domains describe entities in the environment. Jackson distinguishes the domain types *biddable domains* that are usually people, *causal domains* that comply with some physical laws, and *lexical domains* that are data representations. Whenever we have influence on the design of a domain it is a *designed domain*. To describe the problem context, a *connection domain* between two other domains may be necessary. Connection domains establish a connection between other domains by means of technical devices. Examples are video cameras, sensors, or networks. Note that one domain can have more than one type, for example a domain can be a connection and causal domain at the same time.

Interfaces connect domains, and they contain *shared phenomena*. Shared phenomena may be events, operation calls, messages, and the like. They are observable by at least two domains, but controlled by only one domain, as indicated by the abbreviation of that domain and "!". For example, the shared phenomenon *MeterData* in Figure 1 is observable by the domains *SMGSubmitter* and *PersistentMeterData*, but controlled only by the domain *PersistentMeterData* (abbreviation PMD).

[2] http://www.nessos-project.eu/
[3] http://www.openmeter.com/

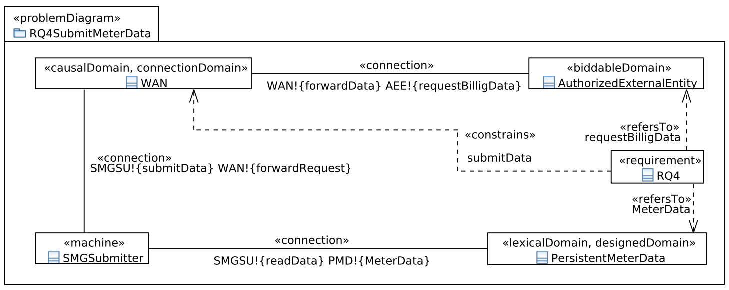

Fig. 1: Problem Diagram `RQ 4` : Submit Meter Data

The objective is to construct a *machine* (i.e., software) that controls the behavior of the environment (in which it is integrated) in accordance with the requirements. Problem-oriented requirements analysis relies on a decomposition of the overall problem into sub-problems, which are represented by *problem diagrams*. Problem diagrams contain the requirements belonging to the sub-problem. When we state a requirement, we want to change something in the environment. Therefore, each requirement *constrains* at least one domain in the environment. A requirement may also *refer* to several domains in the environment of the machine.

The problem frames approach distinguishes between the *requirements (R)*, the *domain knowledge (D)*, and the *specification (S)*. The requirements describe the desired system after the machine is built. The domain knowledge represents the relevant parts of the problem world. The specifications describe the behavior of the software in order to meet the requirements.

We describe problem frames using UML class diagrams, extended by stereotypes, as proposed by Hatebur and Heisel [12]. Figure 1 shows a problem diagram in UML notation. The biddable domain (UML class with stereotype ≪biddableDomain≫) *Authorized External Entity* controls the *request billing data* phenomenon (Name of the UML association with the stereotype ≪connection≫ between the classes *Authorized External Entity* and *WAN*), which is observed by the causal connection domain *WAN* (UML class with stereotype ≪causalDomain, connectionDomain≫). The *SMGSubmitter* controls the *read data* phenomenon, which is observed by the lexical domain *PersistentMeterData* (UML class with stereotype ≪lexicalDomain≫). Additionally, the *SMGProvider submits the data*. The *Persistent Meter Data* controls the *meter data* it contains. The *WAN forwards the data and commands* it observes. The requirement `RQ 4` (for a textual description see Section 4) constrains the *WAN* and refers to the *Authorized External Entity*, and the *PersistentMeterData*.

4 Running Example: Billing

We chose the use case *Meter Reading for Billing* given in the documents of NESSoS and the open meter project to exemplify our method. This use case is concerned with gathering, processing, and storing meter readings from smart meters for the billing process.

Beside the billing use case, there are 13 use cases described for the minimal features of a smart meter gateway in total, which we all considered for our validation. The functional requirements for this use case are defined as follows:

(`RQ 1`) Receive meter data The smart meter gateway shall receive meter data from smart meters.

(`RQ 2`) Process meter data The smart meter gateway shall process meter data from smart meters.

(`RQ 3`) Store meter data The smart meter gateway shall store meter data from smart meters.

(`RQ 4`) Submit billing data The smart meter gateway shall submit processed meter data to authorized external entities.

(`RQ 5`) Provide consumption data to consumer The smart meter gateway shall provide meter data for consumers for the purpose of checking the consistency of bills.

The problem diagram for `RQ 4` was already shown in Figure 1 and explained in Section 3. Figure 2 shows the problem diagram for `RQ 1`. The causal domain *smart meter* controls the *send data* pehomenon, which is forwarded by the *LMN* and finally observed by the machine domain *SMGReceiver*. The *SMGReceiver* controls the phenomenon *writeTemporaryData*, which stores the received information in the lexical domain *temporary meter data*. Additionally, the *SMGReceiver* can *request data* which is forwarded by the *LMN* to the *smart meter*. The causal connection domain LMN *forwards the data and commands* it observes. The requirement `RQ 1` constrains the *temporary meter data* and refers to the *smart meter*. These two problem diagrams are sufficient to understand the rest of the paper, nevertheless we use all five functional requirements for our method.

Note that we will even simplify this example in the following. We will not elaborate on all security elements but restrict ourselves to one example for each element, for example one asset, to improve the comprehensibility for the reader and for reasons of space. Nevertheless, for the validation we elaborated the full case study in means of 27 requirements and all possible assets and attackers.

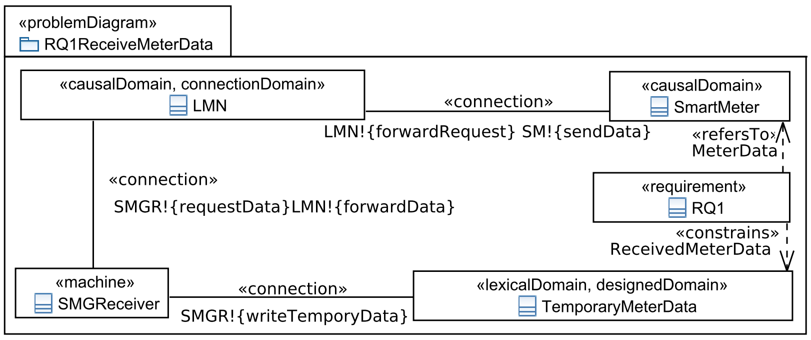

Fig. 2: Problem Diagram `RQ 1`: Receive Meter Data

5 The PresSuRE Method

The PresSuRE method as introduced in [1] consists of four phases and nine steps, which we will briefly explain in the following. For a detailed view we refer the reader to [1].

5.1 Model functional requirements

We assume that the functionality of the system-to-be is described completely, coherently and unambiguously. The functional requirements are a good starting point for a security analysis as the requirements engineer is used to deal with them, they are often already well defined, they already contain everything which has to be protected, and they also contain the entry points for possible attack vectors an adversary can use.

Model Problem Diagrams In the first step of the PresSuRE method, the functional requirements have to be modeled using the *problem frame notation*. This can be done by the *requirements engineer* alone, based on a *textual description* of the *functional requirements*. The result is a set of *problem diagrams* as well as an automatically generated *connection domain discovery table*. *The functional requirements and corresponding problem diagrams are presented in Section 4.*

Adjust Problem Diagrams As setting up problem diagrams allows some degree of freedom, adjustments might be needed to prepare the *problem diagrams*. For the PresSuRE analysis, connection domains are specifically important. But as connection domains are not of central relevance for fulfilling the functional requirements, they are often left out. Hence, one has to make sure that all connection domains are explicitly modeled.

For each connection between domains, the *requirements engineer* and the *system stakeholders* have to check if there is a connection domain in between. The *requirements engineer* and the *system stakeholders* use a table containing the connected domains pairwise, the phenomena in between and a standard questionnaire, which helps to elicit the missing connection domains. For an example of such a table see [1] The result of this step are *adjusted problem diagrams*, which are modeled by the *requirements engineer* using semi-automated wizards. *For our example, using the table and answering the questions, we see that our problem diagrams have to contain WAN, HAN and LMN as connection domains. This information is already reflected by the problem diagrams shown in this paper.*

5.2 Security Knowledge Elicitation

Before starting the security analysis, some security-specific knowledge has to be elicited. This information is crucial for the success of the analysis, as in most cases the functional requirements do not contain enough information for considering security thoroughly. The knowledge about assets in the system-to-be and attackers which might tamper with the system has to be made explicit. As this knowledge is not or only partially available for *requirements engineers*, they have to collaborate with the *stakeholders of the system* and *security experts*.

Prepare Knowledge Elicitation Even though the functional requirements do not contain the information for security analysis, they do already contain some information, which is the starting point for eliciting the additional domain knowledge. We use *security element elicitation tables*, and *attacker elicitation tables* to elicit this information. Examples of such tables are given in [1]. The tables are automatically generated from the *problem diagrams*.

Identify assets, authorized entities and rights The baseline questions for this step are "What has to be protected?" (*asset*), "Who is eligible to access the asset?" (*authorized entities*), and "Which actions are allowed for a stakeholder regarding an asset?" (*rights*). We use the previously generated *security element elicitation tables* to elicit this information. These tables are completed by the *stakeholders of the system-to-be* using the following description, while the *requirements engineer* models the results.

Assets Identify those domains, which have to be protected. Every domain beside the machine is an asset candidate. Most likely one wants to protect a lexical domain representing information or a causal domain. *For our example, we only select the persistent meter data as an asset, which contains information about the electricity consumption of the consumer. This information has to be protected for privacy reasons, as it, for example, allows to monitor the consumer. The full case study contains 13 further assets (see Section 7).*

Authorized Entities An authorized entity to an asset is every domain which has an eligible interest in knowing the state / reading, or controlling / writing the asset. *Eligible entities of the meter data are the smart meters, which produce the meter data, the external entities, who need the consumption information for billing, and the consumer, who wants to check his/her electricity consumption.*

Rights Authorized entities have different rights to access the asset. In case of a lexical domain, the rights are to read or write the information in the domain. In case of the causal domain, the rights are to control or know the state of the causal domain. For each right and authorized entity, one has to state if the entity is allowed to have the right or if the entity must have the right. *The smart meters must have the right to write the information, while the consumer and external entities must have the right to read the information. The smart meters do not need to read the stored consumption data, and the external entities and the consumer are not allowed to modify the consumption data.*

The elicited information has to be added to the model. For this purpose, we use *domain knowledge diagrams*. In domain knowledge diagrams additional knowledge about domains and relations between domains can be modeled. To support modeling security-related domain knowledge we developed UML profiles. The modeling is explained in detail in [1]. The diagrams are generated in the background while the *requirements engineer* completes a wizard which is similar to the security element elicitation table. The result of the step are *asset knowledge diagrams*.

Attacker(s) Elicitation In this step, the *requirements engineer* and a *security expert* have to collaborate to define those *attackers* who might attack our system-to-be. While the *requirements engineer* has a deeper understanding of the system-to-be and its domain, the *security expert* adds his/her vital knowledge about attackers, attacker abilities, possible attack vectors, and so forth. Hence, it is not mandatory that the *requirements engineer* has a security background.

Beckers et al. [13] enumerate different *types of attackers*: *physical attacker*, *software attacker*, *network attacker*, and *social attacker*. Regarding their abilities, we have chosen the abilities as described by Dolev and Yao [14]: *read (read message / get state of domain)*, *write (write message / change state of domain)*, *interfere (intercept message / prevent the change of state)*. For the purpose of eliciting the information about attackers, we use the generated *attacker elicitation tables*.

Attacker First, we have to reason for each domain and type of attacker about the question if this type of attacker might exist for the domain at hand. *For simplicity's sake, we assume for the running example that we only have to defend against network attackers. We distinguish between two network attackers: The internal network attacker, who has access to the HAN and LMN, where the smart meters reside, and the external network attacker, who can attack via the WAN. Note that for the full case study we found and modeled 7 attackers in total, including all kinds of attackers.*

Abilities For each attacker and each domain the attacker has access to, we have to state which abilities the attacker has. Whenever there is no detailed information about the attackers and their abilities regarding a domain they have access to, one should assume the strongest attacker. This might lead to an overestimation of the threats afterward. But adding an unnecessary security requirement is not so much of an issue, while missing one is critical. *After an assessment of all attackers of our example and their abilities, we could not exclude any of the basic abilities. Hence, our attackers have all abilities regarding the domains they have access to.*

The elicited information has to be added to the model to be available for our analysis, too. Again, the modeling can be done semi-automatically using the wizards our tool provides. The result are *attacker knowledge diagrams* (see [1] for more details).

5.3 Graph Generation

The automated part of the security analysis relies on graphs, which visualize information flows and access flows. The attacker asset access graphs, which contain the potential security threats towards the functional requirements, are generated stepwise. The steps and intermediate graphs are explained in the following.

Global Access Graph All graphs $(\mathcal{V}, \mathcal{E})$ that we use for our security analysis in the PresSuRE method are labeled and directed. The set of vertices is a subset of the domains occurring in the model, formally $\mathcal{V} \subseteq Domain$. An edge is annotated with a diagram and a type. The diagram can be a problem diagram or a domain knowledge diagram. The type can be required (*req*), implicit (*imp*) or attack (*att*) ($Type ::= \text{req}|\text{imp}|\text{att}$). The type indicates if the edge is *required* or *implicitly* given by the problem diagram or if it shows a possible *attack* relationship defined in a domain knowledge diagram. The edges point from one domain to another, formally $\mathcal{E} \subseteq Domain \times Diagram \times Type \times Domain$. For the rest of the paper we will regard such an edge as an access flow. In the following, we describe a graph $(\mathcal{V}, \mathcal{E})$ only by its edges $\mathcal{E}$.

For the analysis of the threats towards an asset we will use the *global access graph*. This graph contains the information about access flows between domains, and which problem diagrams are the source of these flows. For the flows, we distinguish between required flows as stated by the requirement and implicit ones which are modeled due to the given environment. To set up the global access graph we use the problem diagrams as an input. The predicates $constrains, refersTo : \mathbb{P}(Domain \times Diagram)$ and $controls : \mathbb{P}(Domain \times Domain \times Diagram)$ can be derived from the problem frame model and are used to generate the global access graph. We have $(d, p) \in constrains$ and $(d, p) \in refersTo$ iff a requirement or domain knowledge in diagram p constrains the domain d or refers to it, respectively. $(d_1, d_2, p) \in controls$ is true iff the domain d_1 controls an interface that d_2 observes in the diagram p.

Using these predicates, we create the global access graph $\mathcal{G}$, which is an overapproximation of the access flows occurring in the system-to-be. An edge $(d_1, p, \text{req}, d_2)$ is in $\mathcal{G}$ iff the domains d_1 and d_2 are not equal, and the domain d_1 is referred to and the domain d_2 is constrained in p. *For example, the problem diagram for* `RQ 1` *(see Figure 2) contains the smart meter and the temporary storage. The smart meter is referred by* `RQ 1` *and the temporary storage is constrained by* `RQ 1`. *Hence, we add a required access flow edge (solid arrow) between smart meter (node with name SmartMeter) and temporary meter data (node with name TemporaryMeterData) annotated with* `RQ 1` *(see graph shown in Figure 3).*

Additionally, an edge $(d_1, p, \text{imp}, d_2)$ is in $\mathcal{G}$ iff the $(d_1, p, \text{req}, d_2)$ is not already in $\mathcal{G}$, the domains d_1 and d_2 are not equal, and d_2 observes an interface controlled by d_1 in p. Note that machines are treated as transitive forwarders in this case. This means that whenever a machine m observes an interface controlled by d_1 and d_2 observes an interface controlled by m, we assume that d_2 observes an interface of d_1. *For example, the domain LMN controls a phenomenon forwardMeterData which is observed by the machine (see Figure 2). The domain temporary meter data observes a phenomenon writeTemporaryData from the machine. Hence, an implicit access flow edge (dotted edge) is added between the LMN and the temporal meter data annotated with* `RQ 1` *(see Figure 3).* The full formal definition is given in [1]

Because of the annotation of the edges we keep the information which problem diagram causes the access flow. Thus, our global access graph contains traceability links that are used in our further analysis. The semantics of an edge $(d_1, p, t, d_2) \in \mathcal{G}$ is that in problem diagram p there is possibly a required or implicit (depending on t) access flow from domain d_1 to domain d_2.

Asset Access Graph As the global access graph can be huge for a complex system-to-be, we introduce an asset access graph which focuses the view on one asset only. It only contains access flows given by the requirements directly or indirectly concerning the asset. Thus, we get one asset access graph per asset. The asset access graph makes

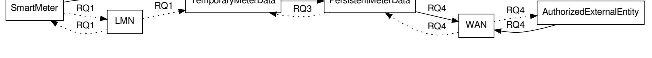

Fig. 3: Global Access Graph (also Asset Access Graph for Persistent Meter Data)

the information for the requirements engineer easier to comprehend. Hence, it improves the scalability of our method. An edge (d_1, p, t, d_2) is in $\mathcal{G}_{asset}$ iff p is in $\mathcal{P}_{access}$. A problem diagram p is in $\mathcal{P}_{access}$ iff there is an edge (d_1, p, t, d_2) which is required and d_1 and d_2 are both in $\mathcal{D}_{access}$. $\mathcal{D}_{access}$ is a union of $\mathcal{D}_{active}$ and $\mathcal{D}_{passive}$. A domain d_1 is in $\mathcal{D}_{active}$ iff there is a required access flow which starts at d_1 and the target domain d_2 is already in $\mathcal{D}_{active}$. Initially, only the $asset$ is in $\mathcal{D}_{active}$. Hence, $\mathcal{D}_{active}$ contains all domains which have a required direct or indirect (via another domain) access flow towards the asset. A domain d_2 is in $\mathcal{D}_{passive}$ iff there is a required access flow which ends at d_2 and the source domain d_1 is already in $\mathcal{D}_{passive}$. Initially, only the $asset$ is in $\mathcal{D}_{passive}$. Hence, $\mathcal{D}_{passive}$ contains all domains which are the target of a required direct or indirect (via another domain) access flow from the asset. The effort for using PresSuRE was reported in detail in [1].

The resulting asset access graph for the persistent meter data is shown in Figure 3, as for our small example the global and the asset access graph do not differ. For a complex scenario the asset access graph is significantly smaller than the global access graph. The asset access graph can be used to check if a stakeholder can gain more rights than he/she should. For reasons of space, we do not go into detail on this matter.

Attacker Asset Access Graph For each asset, we generate the attacker asset access graph, which visualizes the information and control flows from attackers to the asset and from the asset to the attackers. At this point, we focus on the basic information security goals confidentiality, integrity, and availability (short CIA), which are suggested by the Common Criteria [15] and ISO 27000 family of standards [16]. The problematic access flows are annotated with the information which CIA property(ies) are threatened ($CIA ::= \mathrm{C}|\mathrm{I}|\mathrm{A}|\varepsilon$). First, the domains which are directly connected to attackers are identified. Note that for this purpose we use the information given in domain knowledge diagrams created during the step *Identify assets, authorized entities and rights* described in Section 5.2. From these diagrams, we can derive the predicates $read, write, interfere : \mathbb{P}(Domain \times Diagram)$. We have $(d, dk) \in read$, $(d, dk) \in write$, and $(d, dk) \in interfere$ iff domain knowledge in diagram dk has a read, write, or interfere dependency, respectively, to the domain d.

A domain d can be object to be attacked if it is in $\mathcal{D}_{access}$ for the asset at hand. That is, an attacker can access or influence information on the asset through the domain d. We define the sets $\mathcal{D}_w$, $\mathcal{D}_i$, and $\mathcal{D}_r$ as the sets of all domains for which an attacker has the ability to *write*, *interfere*, or *read* it, respectively. A domain d is in $\mathcal{D}_w$ iff there exists an attacker a and a domain knowledge diagram dk, in which d is written and a is referred to by the domain knowledge. The domain d is in $\mathcal{D}_i$ iff there exists an attacker a and a domain knowledge diagram dk in which d is interfered and a is referred as sources of the interference. The domain d is in $\mathcal{D}_r$ iff there exists an attacker a and a domain knowledge diagram dk in which the information in d is referred to and a reads this information. Based on the three sets of domains which might be attacked, the asset threat graph $\mathcal{G}_{threat}$ can be set up. $\mathcal{D}_w$, $\mathcal{D}_i$, and $\mathcal{D}_r$ are formally defined as follows.

$$\begin{aligned}
\mathcal{D}_w = &\{d : \mathcal{D}_{access} \mid \exists a : Attacker; dk : Diagram \bullet (d, dk) \in write \\
&\wedge (a, dk) \in refersTo\} \\
\mathcal{D}_i = &\{d : \mathcal{D}_{access} \mid \exists a : Attacker; dk : Diagram \bullet (d, dk) \in interfere \\
&\wedge (a, dk) \in refersTo\}
\end{aligned}$$

$$\mathcal{D}_r = \{d : \mathcal{D}_{access} \mid \exists a : Attacker; dk : Diagram \bullet (a, dk) \in read \land (d, dk) \in refersTo\}$$

$\mathcal{G}_{threat}$ contains all edges, and therefore problem diagrams, of the corresponding asset access graph which might allow an attacker to successfully attack the asset at hand. An access flow $(d_1, p, t, d_2) \in \mathcal{G}_{asset}$ represents that information is transferred from d_1 to d_2 that possibly comes from the asset or that possibly will be stored in the asset. Hence, such an access flow is a possible threat to the confidentiality of an asset if an attacker has the ability to read one of the domains d_1 or d_2 ($d_1 \in \mathcal{D}_r \lor d_2 \in \mathcal{D}_r$). In this case, we add the edge $(d_1, p, t, \mathrm{C}, d_2)$ to $\mathcal{G}_{threat}$. An access flow $(d_1, p, t, d_2) \in \mathcal{G}_{asset}$ is a possible threat to the integrity of an asset if an attacker has the ability to write the source d_1 of the access flow ($d_1 \in \mathcal{D}_w$), because an attacker could change the information of the asset or the information sent to the asset at domain d_1, which forwards it to domain d_2. In this case, we add the edge $(d_1, p, t, \mathrm{I}, d_2)$ to $\mathcal{G}_{threat}$. We have to consider an access flow $(d_1, p, t, d_2) \in \mathcal{G}_{asset}$ as a possible threat to the availability of an asset if an attacker has the ability to interfere one of the domains d_1 or d_2 ($d_1 \in \mathcal{D}_i \lor d_2 \in \mathcal{D}_i$), because an attacker is then able to threaten the availability of information flowing from or to the asset through the domains d_1 and d_2. In this case, we add the edge $(d_1, p, t, \mathrm{A}, d_2)$ to $\mathcal{G}_{threat}$. $\mathcal{G}_{threat}$ is defined as follows.

$$\begin{aligned}\mathcal{G}_{threat} = \{&(d_1, p, t, cia, d_2) : Domain \times Diagram \times Type \times CIA \times Domain \mid \\ &(d_1, p, t, d_2) \in \mathcal{G}_{asset} \land [(d_1 \in \mathcal{D}_r \lor d_2 \in \mathcal{D}_r) \land cia = \mathrm{C} \\ &\lor d_1 \in \mathcal{D}_w \land cia = \mathrm{I} \lor (d_1 \in \mathcal{D}_i \lor d_2 \in \mathcal{D}_i) \land cia = \mathrm{A}]\}\end{aligned}$$

The full attacker asset access graph $\mathcal{G}_{attack}$ is an extension of $\mathcal{G}_{threat} \subset \mathcal{G}_{attack}$. We add an edge $(d_1, p, t, \varepsilon, d_2)$ to $\mathcal{G}_{attack}$ iff (d_1, dk, t, d_2) is in $\mathcal{G}_{asset}$ but not in $\mathcal{G}_{threat}$. These edges visualize how the attacks on the access flows in $\mathcal{G}_{threat}$ might be propagated over the system due to the functional requirements. Additionally, the attackers are added to the attacker asset access graph. $\mathcal{G}_{attack}$ contains an edge $(a, dk, \mathrm{att}, cia, d)$ if a is an attacker and a domain knowledge diagram dk exists, in which d is referred to and d is written ($cia = \mathrm{I}$) or interfered ($cia = \mathrm{A}$). Additionally, $\mathcal{G}_{attack}$ contains an edge $(a, dk, \mathrm{att}, \mathrm{C}, d)$ if a is an attacker and a domain knowledge diagram dk exists in which d is referred to and a is read. Formally, we define $\mathcal{G}_{attack}$ as follows.

$$\begin{aligned}\mathcal{G}_{attack} = \{&(d_1, p, t, \varepsilon, d_2) : Domain \times Diagram \times Type \times CIA \times Domain \\ &\mid (d_1, p, t, d_2) \in \mathcal{G}_{asset} \land \forall st : CIA \bullet (d_1, p, t, st, d_2) \notin \mathcal{G}_{threat}\} \cup \\ &\{(a, dk, \mathrm{att}, cia, d) : Attacker \times Diagram \times Type \times CIA \times Domain \mid \\ &(d, dk) \in refersTo \land [(a, dk) \in read \land cia = \mathrm{C} \lor (a, dk) \in write \land cia = \mathrm{I} \\ &\lor (a, dk) \in interfere \land cia = \mathrm{A}]\} \cup \mathcal{G}_{threat}\end{aligned}$$

The generated attacker asset access graph for the persistent meter data is shown in Figure 4. Note that for reasons of readability, the PresSuRE tool merges edges and their annotation if they have the same source and target, and are of the same type. The *asset* is now visualized as *ellipse with bold border* and the asset name (*PersistentMeterData*) is written in bold. The *attackers* internal and external network attacker are also added as *ellipses with dashed borders* and in italic font. Their *attack flow edges* are shown as dashed edges, which are annotated with the domain knowledge diagram they are described in and the security goals they may threaten. A bold (both, edge and annotation) access flow indicates a flow for which a security property might be threatened by an

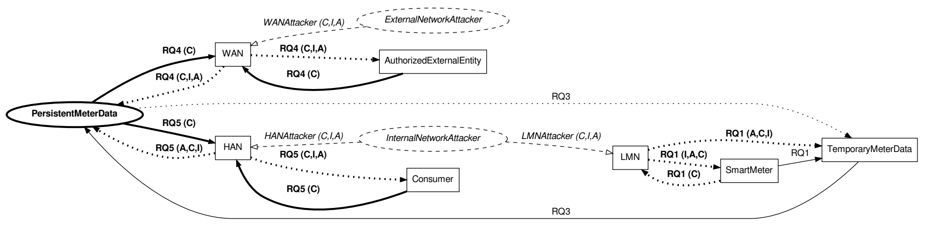

Fig. 4: Attacker Asset Access Graph for Persistent Meter Data

attacker. The threatened security property is annotated in brackets. For example, the implicit access flow edge between the nodes *LMN* and *TemporaryMeterData* is annotated with *RQ1 (A,C,I)*. Hence, it might be possible that for RQ1 the confidentiality, integrity and availability of persistent meter data is threatened.

6 Extending PresSuRE

For the last step of PresSuRE we have to *analyze the attacker asset access graphs* and derive initial security requirements. The input to this step are the attacker asset access graphs. As this step is sparsely described in [1], we elaborate this step and describe the extended the tool support in the following.The attacker asset access graph contains all information regarding access flows to and from the asset at hand. And it contains the information where the asset might be threatened by an attacker. For each asset we identified previously, we check if we have to augment the original requirements related to the asset with security requirements. For each attacker asset access graph, we have to do the following as long as not all problematic access flows are treated:

Select edge First, select a problematic access flow (bold edge with bold annotation) not considered yet. *We select the implicit access flow edge between the nodes LMN and TemporaryMeterData annotated with RQ1 (A,C,I)*

Check confidentiality If there is a (. . . , C, . . .) annotated, we have to check whether there is a threat to the confidentiality of the asset or not. If the threat can occur for the annotated requirement, we have to augment this requirement with a confidentiality requirement. *Indeed, the confidentiality is threatened by internal network attackers. If they are able to learn all data sent by the smart meters, they can derive the information contained in the persistent meter data by themselves. Hence, we have to add a confidentiality requirement complementing RQ1.*

Check integrity If there is an (. . . , I, . . .) annotated, we have to check whether there is a threat to the integrity of the asset or not. If the threat can occur for the annotated requirement, we have to augment this requirement with an integrity requirement. *The integrity is threatened by internal network attackers. If they are able to add data or change data sent by the smart meters, they can change the information contained in the persistent meter data. This is a threat as the persistent meter data is the basis for billing. Hence, we have to add an integrity requirement complementing RQ1.*

Input
domain accessed by the attacker, *the attacker*, and *threatened security property*

Precondition
Precondition 1: Domain accessed by the attacker is a connection domain
Precondition 2: Security property threatened is integrity or confidentiality

Template
Title: Secure access flows via [*domain accessed by the attacker*]
Text: The access flows via the [*domain accessed by the attacker*] must be secured in a way that the [*the attacker*] is not able to threaten the [*threatened security property*] of the access flows.

Table 1: SR Template for Connection Domains, and Integrity and Confidentiality

Check availability If there is an (. . . , A, . . .) annotated, we have to check whether there is a threat to the availability of the asset or not. If the threat can occur for the annotated requirement, we have to augment this requirement with an availability requirement. *The availability is threatened by internal network attackers. If they are able to deny the service of the LMN, no data can be sent by the smart meters. Thus, the persistent meter data cannot be computed and used for billing. Hence, we have to add an availability requirement complementing RQ1.*

The iteration over the assets, and the iteration over the edges in an according attacker asset access graph for the asset at hand, is guided by the tool. It indicates the asset and the edge in question and shows the according attacker asset access graph. The requirements engineer and security expert have to do the reasoning and provide the result to the tool. From this information we collected for an edge, we can derive initial security requirements. The initial security requirements can be generated automatically, using templates. For example, the template for a security requirement regarding a connection domain which can be accessed by an attacker to threaten the security properties confidentiality and integrity is shown in Table 1. Such a template defines the inputs for filling the templates. In this case, we need the attacker, the domain he/she can access and the security property threatened by the access. To instantiate the template in a reasonable way some preconditions must be fulfilled. First, the domain accessed by the attacker must be a connection domain. Second, the security property threatened must be integrity or confidentiality. The template itself is given as gap-text in which the gaps are indicated by brackets. Within a bracket the input element is referenced which will later on replace the bracket when instantiating the template. Such a template also contains the modeling rules to add the security requirement to the problem frames model. For sake of brevity we do not show and discuss these rules in detail. An example model is shown in Fig. 5. In general, we stick the profile and rules as defined in [17].

The templates for the different cases are implemented in the tool. *Hence, for our example we can generate the following requirement regarding confidentiality:* ***SRQ 1.1 Secure access flows via*** **LMN** *The access flows via the* LMN *must be secured in a way that the* InternalNetworkAttacker *is not able to threaten the* confidentiality *of the access flows. Figure 5 shows the according modeling, in which the confidentiality requirement SRQ1.1 (UML class with stereotypes ≪requirement, confidentiality≫) complements (UML dependency with stereotype ≪complements≫)* `RQ 1` *(UML class with stereotype ≪requirement≫). SRQ1.1 constrains (UML dependency with stereotype ≪constrains≫) the LMN (UML class with stereotypes ≪connectionDomain,*

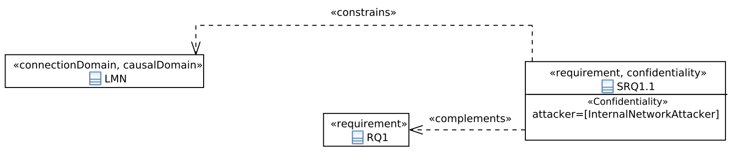
«constrains»
«connectionDomain, causalDomain» LMN
«requirement, confidentiality» SRQ1.1
«Confidentiality» attacker=[InternalNetworkAttacker]
«requirement» RQ1
«complements»

Fig. 5: Snipped from the Problem Diagram for `RQ 1` Augmented with SRQ1.1

causalDomain$\gg$). SRQ1.1 considers the InternalNetworkAttacker (Property attackers of SRQ1.1). We treat the integrity and availability threat for the selected edge in the same way.

Every newly added security requirement has an impact on the attacker asset access graph at hand. But but it also has an impact on other attacker asset access graphs whenever an attacker asset access graph contains edges, which appear due to the functional requirement that is complemented by the newly added security requirement. Hence, it is necessary to reduce all attacker asset access graphs to ensure that one only analyzes edges which are not treated already by an security requirement. For the specification of the reduction of attacker asset access graphs, we need two additional predicates The predicate $complements : \mathbb{P}(Requirement \times CIA \times Attacker)$ can be derived from the problem frame model. We have $(r, cia, a) \in complements$ iff a security requirement of the security property cia, which refers to attacker a, complements the requirement r. The predicate $models : \mathbb{P}(Requirement \times Diagram)$ can be derived from the problem frame model. We have $(r, p) \in models$ iff a requirement r is part or the diagram p. Additionally, we need the set $\mathcal{R}_{access}$. $\mathcal{R}_{access}$ contains the tuples $(r, cia, a) : Requirement \times CIA \times Attacker$ which relate the requirement r to the attacker a who exploits r to threaten the security property cia. A tuple (r, cia, a) is in $\mathcal{R}_{access}$ iff there is an access flow (d_1, p, t, cia, d_2) part of the attacker asset access graph G_{attack} for which the requirement r is part of the diagram p and if $\mathcal{G}_{attack}$ contains an edge $(a, dk, \text{att}, cia, d_1)$ or an edge $(a, dk, \text{att}, cia, d_2)$.

$$\begin{aligned}\mathcal{R}_{access} =& \{(r, cia, a) : Requirement \times CIA \times Attacker \mid \exists (d_1, p, t, cia, d_2) \in \mathcal{G}_{attack} \bullet \\ & r \in p \wedge (\exists (a, dk, \text{att}, cia, d_1) \in \mathcal{G}_{attack} \vee \exists (a, dk, \text{att}, cia, d_2) \in \mathcal{G}_{attack}\}\end{aligned}$$

Based on $\mathcal{R}_{access}$ and $\mathcal{G}_{threatOld}$, which is equal to $\mathcal{G}_{threat}$ calculated before we introduce a new security requirement, we can now update $\mathcal{G}_{threat}$. $\mathcal{G}_{threat}$ now contains all edges, and therefore problem diagrams, of the corresponding asset access graph which might allow an attacker to successfully attack the asset at hand and this attack isnot mitigated by an according security requirement. An access flow $(d_1, p, t, cia, d_2) \in \mathcal{G}_{threatOld}$ is also contained in $\mathcal{G}_{threat}$ iff there exists an requirement r and an attacker a for which the requirement r is modeled in the diagram p and the requirement r enables the attacker a to threaten cia and this access is still not mitigated by a complementing security requirement. Formally, we define the new $\mathcal{G}_{threat}$ as follows:

$$\begin{aligned}&\textbf{required } \mathcal{G}_{threatOld} \\ &\mathcal{G}_{threat} = \{(d_1, p, t, cia, d_2) : Domain \times Diagram \times Type \times CIA \times Domain \mid \\ &\qquad (d_1, p, t, cia, d_2) \in G_{threatOld} \wedge (\exists r : Requirement, a : Attacker \bullet \\ &\qquad (r, p) \in models \wedge (r, cia, a) \in \mathcal{R}_{access} \setminus complements)\}\end{aligned}$$

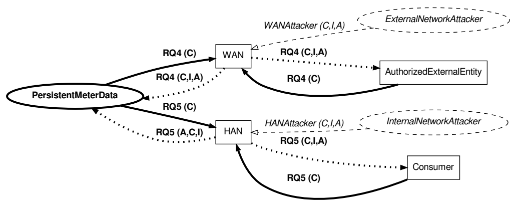

Fig. 6: Attacker Asset Access Graph for Persistent Meter Data After Reduction

The updated threat graph $\mathcal{G}_{threat}$ leads to an updated and reduced attacker asset access graph $\mathcal{G}_{attack}$. Hence, the tool can ensure that only edges are analyzed which are not already treated. Additionally, the tool is now able to detect that an asset is not threatened anymore as $\mathcal{G}_{attack}$ is gradually reduced till it is empty. *As we have added security requirements for integrity, confidentiality, and availability complementing* `RQ 1`*, the attacker asset access graphs can be reduced now. Figure 6 shows the attacker asset access graph for the persistent meter data after the reduction (the initial graph is shown in Fig 4). Now the smart meters, the LMN, and the temporary meter data are not part of the graph, as the threats which made them relevant are already considered.*

7 Validation

We validated PresSuRE using two real-life case studies, the already introduced smart meter and a voting system. The voting system requirements were obtained from a Common Criteria profile for voting systems [18]. For more details on the voting system case study, see [19]. The results for applying PresSuRE are reported in the following in detail for the Smart Grid. The original functional requirements were obtained from [11] and the NESSoS case studies provided by the industrial partners of the project. For conducting our method, we selected 13 minimum uses cases, which embody 27 requirements in total. For these requirements, 14 assets and 7 attackers of all kinds, as described in Section 5.2, were identified. Based on this information, the graphs were generated, and the initial security requirements elicited.

We analyzed each attacker asset access graph for assessing the tool support, and we also analyzed the initial security requirements found for assessing the overall method.

Table 2: Results of the assessment for the smart meter case study

	Confidentiality	**Integrity**	**Availability**
Precision per attack asset access edges	36.14%	66.35%	62.52%
Recall per attack asset access edges	100.00%	100.00%	100.00%
Aggregated precision per attack asset access edges	55.00%		
Aggregated recall per attack asset access edges	100.00%		
Precision per requirement	92.59%	96.30%	96.30%
Recall per requirement	100.00%	100.00%	100.00%
Aggregated precision per requirement	95.06%		
Aggregated recall per requirement	100.00%		

For the graph, we checked for each edge in the attacker asset access graph at hand if the annotated threats are existing according to the threats and security requirements of the original documents (e.g. [11,10] for smart meter). We also looked for threats and security requirements which are defined in such documents, but which were not identified using PresSuRE. In this way we were able to measure the precision and recall of our method. Unfortunately, we do not know which security analysis was used for eliciting the security requirements reported in those documents. But we assume that security experts were involved in writing the documents and the documents were reviewed thoroughly. Hence, these documents are a good benchmark.

Next, we aggregated the results of the edges of the attacker asset access graph for each requirement. Thus, we derived for each requirement the information if the requirement has to be complemented by security requirements according to PresSuRE. Again, we also checked if the found security requirements are compliant with the original documents. Last, we measured the precision and recall of PresSuRE on the requirements level.

The results of this analysis for the smart meter is shown in Table 2. Speaking of the precision on the level of edges of the attacker asset access graph, we have many false positives, especially for confidentiality. This is because the original documents do not demand a high level of confidentiality. Additionally, PresSuRE discovered potential indirect information flows between assets which will not occur in the system later on. Thus, PresSuRE is very strict and defensive, which is not appropriate in every case. Note that even though the indirect flows often turned out to be irrelevant, they have to be checked anyway. Often attacks use such indirect relations to tamper with a system. Overall, the precision on the level of edges of the attacker asset access graph is acceptable (55%), but should be improved. The recall is perfect (100%) as we did not find any false negatives. On the requirements level, our results are satisfying. Whenever PresSuRE suggested to add a complementing security requirement for a functional requirement, this suggestion was correct with a precision of 95%, and no security requirment was missed (recall 100%).

Similar results were obtained for the voting system case study. The precision on the level of edges of the attacker asset graph is slightly higher, as the voting system documents are very strict regarding confidentiality. This fact is also reflected on the requirements level, but the difference to the smart meter case study is not significant. For the attacker asset access edge and the requirements level the recall was 100% again.

Speaking of the effort, we spent 43 person hours, which is a significant effort, but seems to be reasonable.The effort for using PresSuRE was reported and discussed in detail in [1].

8 Related Work

Schmidt and Jürjens [20] propose to integrate the SEPP method, which is based on problem frames, and UMLSec [21], which is based on a UML profile and allows tool-based reasoning about security properties. In this way, they can express and refine security requirements and transfer the security requirements to subsequent design artifacts. A similar method is described by Haley et al. [22], which also relies on problem frames for security requirements analysis. The first method [20] starts after the initial security requirements are already known, while the latter one already embodies a step for security requirements elicitation. But this particular step is described very sparsely and informally. Hence, our work can complement and improve these works.

There are many publications concerning goal-oriented security requirements analysis (e.g. [23,24,25,26]). But goal models are of a higher level of abstraction than problem frames. Goal models are stakeholder-centric, while problem frames are system-centric. Therefore, refining functional requirements taking into account more detail of the system-to-be and analyzing the system-to-be described by the functional requirements is reported to be difficult for goal-oriented methods [27]. Alrajeh et al. try to tackle this problem by introducing refinement steps which rely on heavy weight formalizations. We offer an alternative way of bridging the this gap. Thus, even though the goals of an attacker and their implication for the goals of stakeholders are already known, one might benefit from using our method.

9 Conclusion

In this paper, we extended a methodology for Problem-based Security Requirements Elicitation (PresSuRE). PresSuRE is a method for identifying security needs during the requirements analysis of software systems using a problem frame model. Our extension now enables a guided analysis of found threats. In consequence, security requirements can be derived in a structured way. In summary, the PresSuRE method extension has the following advantages: It introduces a method to reduce attacker asset access graphs successively by adding security requirement, which 1) allows to visualize the impact of an security requirement on the attacker asset access graphs, 2) visualizes the unmitigated threats, and 3) avoids analysis of threats which are not relevant anymore. And it is a re-usable requirements security analysis method which 1) relies on only single access flow which the analyst can be easily comprehend, 2) allows to derive security requirements in a structured way, 3) eases the formulation and modeling of the security requirements, 4) is applicable to different domains, and 5) is tool supported to ease analysis, and modeling necessary for the method.

We validated our method and tool with two real-life case studies in the fields of smart grid and voting systems. The results show the suitability of our method to detect initial security requirements. For the future, we plan to investigate how the basic CIA properties can be refined further systematically into more fine-grained security requirements such as authentification and authorization.

References

1. Faßbender, S., Heisel, M., Meis, R.: Functional requirements under security pressure. In: ICSOFT-PT 2014 - Proceedings of the 9th International Conference on Software Paradigm Trends, Vienna, Austria, 29-31 August, 2014. (2014)
2. Bundeskriminalamt (federal criminal police office): Bundeslagebild Cybercrime 2013 (report on cybercrime 2013). Technical report, Germany (2014)
3. Bundeskriminalamt (federal criminal police office): Bundeslagebild Cybercrime 2012 (report on cybercrime 2012). Technical report, Germany (2013)

4. Norton: Norton Report 2013. Technical report, Norton (2013)
5. Willis, R.: Hughes Aircraft's Widespread Deployment of a Continuously Improving Software Process. AD-a358 993. Carnegie-mellon university (1998)
6. Boehm, B.W., Papaccio, P.N.: Understanding and controlling software costs. IEEE Transactions on Software Engineering **14**(10) (1988) 1462–1477
7. Firesmith, D.: Specifying good requirements. Journal of Object Technology **2**(4) (2003)
8. Beckers, K., Faßbender, S., Heisel, M., Meis, R.: A problem-based approach for computer aided privacy threat identification. In: APF '12, Springer (2013) 1–16
9. Jackson, M.: Problem Frames. Analyzing and structuring software development problems. Addison-Wesley (2001)
10. Kreutzmann, H., Vollmer, S., Tekampe, N., Abromeit, A.: Protection profile for the gateway of a smart metering system. Technical report, BSI (2011)
11. : Requirements of AMI. Technical report, OPEN meter project (2009)
12. Hatebur, D., Heisel, M.: Making pattern- and model-based software development more rigorous. In: ICFEM '10, Springer (2010) 253–269
13. Beckers, K., Hatebur, D., Heisel, M.: A problem-based threat analysis in compliance with common criteria. In: ARES '13, IEEE Computer Society (2013)
14. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2) (1983) 198–207
15. ISO/IEC: Common Criteria for Information Technology Security Evaluation. ISO/IEC 15408, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), Geneva ,Switzerland (2009)
16. ISO/IEC: Information technology - Security techniques - Information security management systems - Overview and Vocabulary. ISO/IEC 27000, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), Geneva ,Switzerland (2009)
17. Hatebur, D., Heisel, M.: A UML Profile for Requirements Analysis of Dependable Software. In Schoitsch, E., ed.: Proc. of the Int. Conf. on Computer Safety, Reliability and Security (SAFECOMP). LNCS 6351, Springer (2010) 317–331
18. Volkamer, M., Vogt, R.: Common Criteria Protection Profile for Basic set of security requirements for Online Voting Products. Bundesamt f"ur Sicherheit in der Informationstechnik. (April 2008)
19. Faßbender, S., Heisel, M.: From problems to laws in requirements engineering using model-transformation. In: ICSOFT '13, SciTePress (2013) 447–458
20. Schmidt, H., Jürjens, J.: Connecting security requirements analysis and secure design using patterns and UMLsec. In: CAiSE '11, Springer (2011) 367–382
21. Jürjens, J.: Secure Systems Development with UML. Springer (2005)
22. Haley, C.B., Laney, R., Moffett, J.D., Nuseibeh, B.: Security requirements engineering: A framework for representation and analysis. IEEE Transactions on Software Engineering **34**(1) (2008) 133–153
23. Liu, L., Yu, E., Mylopoulos, J.: Security and privacy requirements analysis within a social setting. In: RE '03. (2003) 151–161
24. Mouratidis, H., Giorgini, P.: Secure Tropos: a security-oriented extension of the tropos methodology. International Journal of Software Engineering and Knowledge Engineering **17**(2) (2007) 285–309
25. Salehie, M., Pasquale, L., Omoronyia, I., Ali, R., Nuseibeh, B.: Requirements-driven adaptive security: Protecting variable assets at runtime. In: RE '12. (2012) 111–120
26. Van Lamsweerde, A.: Elaborating security requirements by construction of intentional anti-models. In: ICSE '04. (2004) 148–157
27. Alrajeh, D., Kramer, J., Russo, A., Uchitel, S.: Learning operational requirements from goal models. In: ICSE '09. (2009) 265–275