# A Systematic Method to Describe and Identify Security Threats Based on Functional Requirements

Roman Wirtz(✉) and Maritta Heisel

University of Duisburg-Essen, Duisburg, Germany
{roman.wirtz,maritta.heisel}@uni-due.de

**Abstract.** Scenarios in which the security of software-based systems is harmed become more and more frequent. Such scenarios can lead to substantial damage, not only financially, but also in terms of loss of reputation. Hence, it is important to consider those threats to security already in the early stages of software development. However, it is non-trivial to identify all of them in a systematic manner. In particular, the knowledge about threats is not documented in a consistent manner. The *Common Vulnerability Scoring System* is a well known way to characterize vulnerabilities in a structured way. Our idea is to document threats in a similar way, using a template. A distinguishing feature of our approach is that we relate the threats to the envisaged functionality of the software. Our contribution is two-fold: first, we propose a general template to describe security threats that can be used in the early stages of software development. Second, we define a systematic and semi-automatic procedure to identify relevant threats for a software development project, taking the functionality of the software-to-be into account.

## 1 Introduction

In the last few years, the number of documented security incidents highly increased. Such an incident causes value and reputation loss. Additionally, fixing vulnerabilities leading to those incidents is cost intensive. A risk for a software can be defined as the combination of the likelihood of incidents and the consequence for an asset due to that incident. A risk management process defines different steps for coordinating activities to reduce the risk. The challenge for software engineers is to identify and control risks as early as possible. Following the principle of security-by-design, software is built from the beginning with regard to security.

Most knowledge about threats that might lead to a harm for assets exists for already running software. An example of such a knowledge base is the *National Vulnerability Database (NVD)*[1]. In these knowledge bases, it is possible to search for vulnerabilities based on component names like *MySQL Server*. In the analysis

[1] NVD - https://nvd.nist.gov/ (accessed on 2018-02-01).

© Springer Nature Switzerland AG 2019
A. Zemmari et al. (Eds.): CRiSIS 2018, LNCS 11391, pp. 205–221, 2019.
https://doi.org/10.1007/978-3-030-12143-3_17

phase of a software development project, however, we need a more abstract view on vulnerabilities, which is for example provided by the *Open Web Application Security Project* [1]. Such abstract resources, like OWASP, are focused on specific domains, e.g. web applications, and often do not provide a common structure to characterize possible threats. We aim to assist software engineers in managing risks based on the functional requirements of software systems.

In this paper, we contribute to the identification of risks. Our first contribution is a template to describe threats independently of the application domain, specific design decisions, or implementation details. The template is based on the *Common Vulnerability Scoring System* [2]. To integrate threats into models of functional requirements, we propose a new diagram type based on *Problem Frames* [3]. The extended model later supports security engineers in defining security requirements for the software to be developed. A method to identify relevant threats based on the template is our second contribution. The method is semi-automatic, which means, that we want to minimize the manual interaction for security engineers to perform the method as most as possible. Both, the template and the method are applied in the early stages of software development to ensure security-by-design.

The paper is structured as follows: In Sect. 2, we briefly summarize the concepts on which our work is based. A conceptual model describing the used terminology is given in Sect. 3. Section 4 introduces a general template to describe threats. The template is exemplified by an instantiation for a sample threat. The identification method for relevant threats is explained in Sect. 5. Related work is discussed in Sect. 6. We conclude our work with a discussion and future work in Sect. 7.

## 2 Background

In this section, we introduce the fundamentals on which our work is based. First, we present Michael Jackson's problem frames approach [3] to model functional requirements. Afterwards, we introduce the ProCOR method [4] which is a problem-based risk management process.

### 2.1 Problem Frames

To model requirements, we make use of Michael Jackson's problem frames approach [3]. Problem frames are patterns to characterize subproblems of a complex software development problem. These patterns are used in the early stages of the software development life-cycle. An instance of such a pattern is called problem diagram (examples are given in Figs. 3, 4 and 5). It contains a functional requirement **FR** (dashed ovals) for the system-to-be. A requirement is an optative statement which describes how the environment should behave when the software is installed. The entities related to a requirement are represented as domains (gray rectangles). There are different types of domains: biddable domains **B** (e.g., persons), causal domains **C** (e.g., technical equipment), machine domains **M** (representing the piece of software to be developed), lexical domains **X** (data representations) and display domains **D** (visual output devices). There are symbolic phenomena, representing some kind of information or a state, and causal phenomena, representing events, actions and so on. Each phenomenon is controlled by exactly one domain and can be observed by other domains. A phenomenon controlled by one domain and observed by another is called a shared phenomenon between these two domains. Interfaces (solid lines) contain sets of shared phenomena. Such a set contains phenomena controlled by the same domain (indicated by $A!\{...\}$, where $A$ is an abbreviation for the controlling domain). Some phenomena are *referred to* by a requirement (dashed line to the controlling domain), and at least one phenomenon is *constrained* by a requirement (dashed lines with arrowhead to the controlling domain). The domains and their phenomena that are referred to by a requirement are not influenced by the machine, whereas we build the machine to influence the constrained domain's phenomena in such a way that the requirement is fulfilled.

Faßbender et al. [5] describe a method to combine aspects with problem frames. Aspects are considered as cross-cutting concerns for different requirements. To integrate aspects into problem frames, join points are defined. A join point, is a placeholder in an aspect diagram. We mark those join points in white, whereas normal domains are given in gray (cf. Table 2). By mapping the join points to domains of the problem frame, the aspects are integrated into the problem frame. Mapping means that a join point is replaced by the corresponding domain contained in the problem diagram in which the aspect shall be integrated.

## 2.2 ProCOR

Wirtz et al. [4] propose a risk- and problem-based method to identify security requirements in the early stages of a software development process. According to the principles of risk management described in ISO 31000 [6], the ProCOR method contains steps for risk identification, risk evaluation and risk treatment. The step for risk identification is described as a structured brainstorming with experts. The template to describe threats and the identification method we propose in this paper assists the security engineer in risk identification step.

In ProCOR, an asset is defined as some kind of information to be protected with regard to a security goal (confidentiality, integrity or availability). A piece of information is described as a phenomenon used in a problem diagram. The set of assets is the focus of the analysis. As the scope of the analysis we consider all domains at which some information to be protected is available. Available means that a domain controls or observes phenomena in which the information is contained. ProCOR defines a so-called information flow graph to derive these domains automatically, based on a set of problem diagrams. The identification of relevant threats described in this paper is based on this scope definition.

## 3 Conceptual Model

We first introduce a conceptual model, we developed in previous work based on the ISO 27005 standard [7]. The model is shown in Fig. 1 and describes the terminology we make use of in this paper. The ISO 27005 standard [8] has a special focus on security and is based on the ISO 31000 standard.

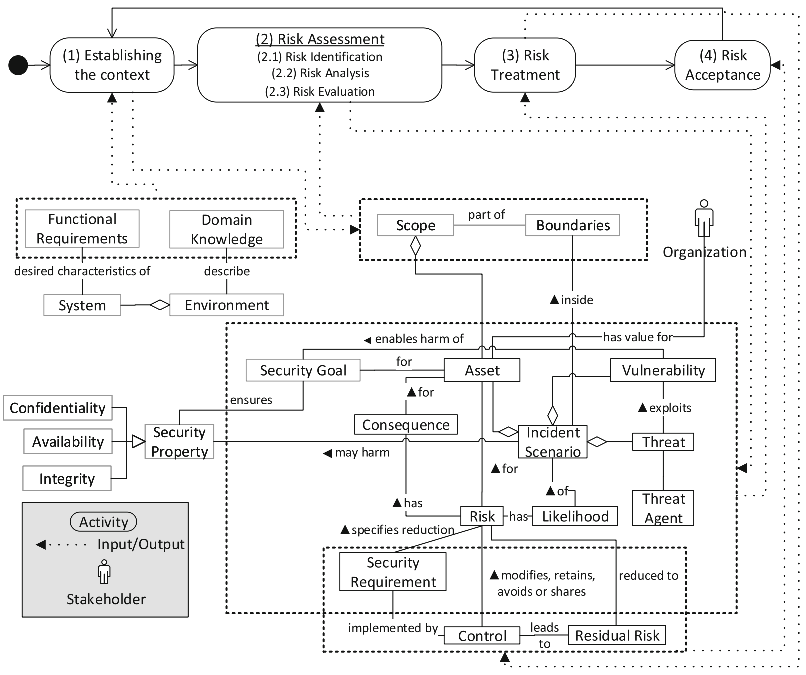


**Fig. 1.** Conceptual model [7]

According to the standard, a risk management process consists of four steps: (1) Establishing the context, (2) Risk Assessment, (3) Risk Treatment, and (4) Risk Acceptance. Risk Assessment contains three sub activities: (2.1) Risk Identification, (2.2) Risk Analysis and (2.3) Risk Evaluation. In this paper, we contribute to the identification of risks. The initial input is a set of *Functional Requirements* which describe the desired characteristics of the *System*. The system is part of the *Environment* which is described by *Domain Knowledge*. An *Asset* is part of the *Scope* and the scope is inside the *Boundaries* of the analysis. An asset has some value for an *Organization*. In the following, we consider a piece of information as an asset. For an asset, there is a *Security Goal* which ensures a *Security Property* for an asset. We consider the following types of properties: (1) *Confidentiality*: Some piece of information shall not be disclosed to unauthorized third parties. (2) *Integrity*: Some piece of information shall not be altered by unauthorized third parties. (3) *Availability*: Some piece of information shall be available for authorized parties. A security goal might be harmed when a *Threat* exploits a *Vulnerability*. According to ISO 27005, a threat might be human or natural (for instance a thunderstorm), and deliberate or accidental. In the Common Criteria [9], a *Threat Agent* is explicitly mentioned. A threat is the action which has a negative influence on assets, and a threat agent performs that action. An *Incident Scenario* includes a threat, a vulnerability and an asset that might be harmed. Such a scenario has a likelihood which is further used to evaluate risks. A *Risk* is related to that likelihood and the specific consequence for the asset. A risk can be treated (e.g. modified, retained, avoided or shared) by some *Control*s. After applying controls, there is still a *Residual Risk*, because usually, a risk cannot be eliminated. The risk reduction is specified by a *Security Requirement* which is implemented by some controls. In the following, we focus on the description of threats. A detailed description of threats is essential for the risk evaluation and the treatment of risks by selecting appropriate treatments.

## 4 Threat Description

In this section, we introduce a template to describe threats. We also describe how to integrate these threats into the model of functional requirements. This enables developers to consider threats along with functional requirements during the software development life-cycle.

### 4.1 Template Format

The structure of the template is inspired by the *Common Vulnerability Scoring System (CVSS)* [2]. The CVSS is used to score known vulnerabilities. This scoring system enables the security engineers to calculate the severity of a discovered vulnerability based on attacker information, information about the consequences and how the vulnerability can be used. Usually, the CVSS is used to classify vulnerabilities according to the CVSS User Guide [10].

In Tables 1 and 2, we show the structure of the template for an example threat. A threat is characterized by different attributes. The left-hand column states the name of an attribute, and the right-hand column shows its corresponding value. We extend the structure of the CVSS with new attributes (marked gray). There are four sections: *Basic Information*, *Threat Information*, *Relevant Problem Diagrams* and *Integration*.

**Basic Information.** The attributes contained in the section *Basic Information* are used to provide an overview of the described threat. There is a *name*, a *context* in which the threat might occur, a *reference* to the original document that describes the threat, a list of *keywords*, and an informal *description* of the threat. The *Vulnerability* that enables the harm of the security goal and the *Consequences* are described informally.

**Table 1.** Threat description: injection part 1

| **Basic Information** | |
|---|---|
| Name | *Injection* |
| Context | *Application that provides some user input to select or edit some data.* |
| Reference | *OWASP Top Ten 2017 [1]* |
| Keywords | *injection, database, untrusted data* |
| Description | *Data entered by users is not validated and used in queries to read or modify data, e.g. SQL queries. An attacker needs to be able to input data which is then used to query or modify data.* |
| Vulnerability | *User input is not validated before execution.* |
| Consequences | *Data is manipulated, deleted or disclosed by unauthorized persons.* |
| **Threat Information** | |
| Threat Type | ☐ Accidental ☑ Deliberate |
| Threat Agent | ☑ Human ☐ Technical ☐ Natural |
| Threat Vector | ☑ Network ☑ Adjacent ☑ Local ☐ Physical ☐ Personal (Social Engineering) |
| Type of affected domain | ☐ causal ☑ machine ☐ lexical ☐ biddable ☐ display |
| Type of target domains | ☐ causal ☐ machine ☑ lexical ☐ biddable ☐ Display |
| Complexity | ☑ Low ☐ High |
| Privileges Required | ☐ None ☑ Low ☐ High |
| User Interaction | ☑ None ☐ Required |
| Threat Scope | ☐ Unchanged ☑ Changed |
| Confidentiality Impact | ☐ None ☐ Low ☑ High |
| Integrity Impact | ☐ None ☐ Low ☑ High |
| Availability Impact | ☐ None ☐ Low ☑ High |

**Threat Information.** The section *Threat information* is divided into several attributes describing the characteristics of the threat. The CVSS concentrates on attacks, but we consider threats in general. The attribute *Threat Type* indicates whether a threat is accidental or deliberate. A *Threat Agent* might be human, technical or natural.

The *Threat Vector* describes the possible ways of accessing the piece of software to realize the threat. In the CVSS specification, it is called Attack Vector. *Network* describes threats that can be realized from any network, for example a wide area network, *adjacent* stands for local network access, *local* means that the threat agent needs direct access to the computer, and *physical* describes physical access to components such as a hard disk. To describe the communication between humans, for example by bribing an employee, we add the value *Personal (Social Engineering)*.

To map the threat description to the initial set of functional requirements, we also document the types of related domains. The *type of affected domain* documents the type of the domain on which the threat takes place. This domain might differ from the domains on which the harmed asset is available. For instance, attacking software (machine domain) might be necessary to get access to some data (lexical domain). Hence, we need to document the *type of the target domains*, as well.

The *Complexity* has two qualitative values: *low* and *high*. A low complexity means that a threat agent can expect repeatable success when realizing the threat without specialized access conditions or preparations. In contrast to that, a high complexity is considered when the threat agent has to invest some measurable effort in preparing the action that might harm the asset. For instance, an attacker needs local access to the server by breaking into the server room.

There are three possible values to state whether privileges are required. *None* means that no special privileges are required, *low* stands for a user account, and *high* means that administrator rights are necessary to realize the threat.

In some cases, an additional user interaction is necessary to realize the threat, for example an end-user has to confirm the installation of additional software. This is indicated by the corresponding attribute.

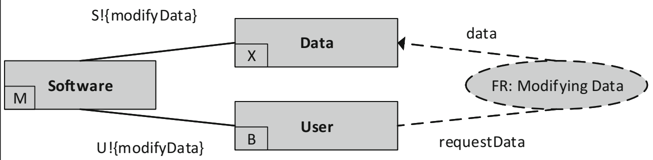
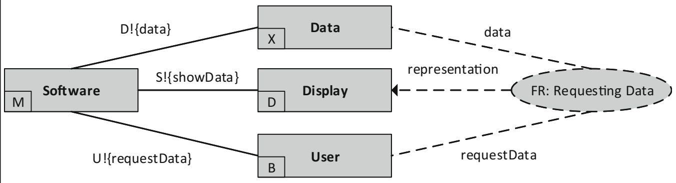
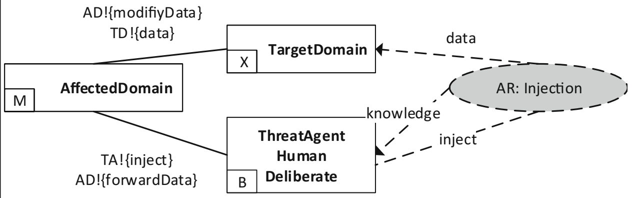
The initial *Scope* of a threat is the domain on which the threat is realized. In case that the attacker gets only access to this domain, the scope remains *unchanged*. If the threat affects additional domains, the scope is considered as *changed*, i.e. the target domain differs from the affected domain.

For the security goals confidentiality, integrity and availability, a qualitative scale for the impact on the security goal is defined. *None* means that there is no impact on this security goal, *Low* means access or manipulation only to parts of the information to be protected and *High* means full access or at least major impact to the information to be protected. However, it is hard to define the impact independently of the concrete application. Hence, the given scale can only be considered as a minor indicator for the risk evaluation.

**Relevant Problem Diagrams.** It does not suffice to consider the aforementioned attributes to decide whether a threat is relevant or not. Some threats require a specific functionality to be realized. For example, to perform an injection, user input is required. Without such an input, the threat is not relevant. The functional requirements of a software system can be described with problem diagrams. To relate threats to functional requirements, we introduce the section *Relevant Problem Diagrams* as shown in Table 2. The given diagrams describe the minimal set of elements to be contained in a problem diagram for which the threat might be relevant. In the left-hand column, we annotate the corresponding consequence. The graphical representation is shown in the right-hand column.

**Integration.** The section *Integration* describes how the threat can be integrated into the model of functional requirements. We follow an aspect-oriented approach (see Sect. 2.1) to consider threats along with functional requirements. Lin et al. [11] introduce the notion of an anti-requirement (**AR**) to describe the undesired behavior of software due to an attack. We make use of anti-requirements to describe the system's behavior when a threat becomes effective. We consider an anti-requirement as a special type of aspect, because the relevance of a threat is not limited to one specific functional requirement. We introduce a new kind of aspect diagram, called *Threat diagram*. Such a diagram is part of the threat description. It is used to identify the interfaces and domains that are related to the described threat. The anti-requirement constrains the undesired phenomena, for example the information flow from a domain to a threat agent. It refers to the phenomena which enable the threat, for example the injection performed by an attacker. In the further steps of a risk management process, security engineers have to ensure that the anti-requirement cannot be fulfilled by choosing appropriate controls. Such controls have to be selected with regard to the functional requirements, which means that treatments shall not interfere with the functionality. By combining threats and functional requirements in one model, we provide a global view for security engineers on the functionalities and threats and help to investigate both aspects in a whole.

**Table 2.** Threat description: injection part 2

| **Relevant Problem Diagrams** | |
|---|---|
| **Consequences:** Manipulation, Deletion |   S!{modifyData} Data X Software M User B U!{modifyData} data FR: Modifying Data requestData |
| **Consequence:** Disclosure |   D!{data} Data X Software M S!{showData} Display D User B U!{requestData} data representation FR: Requesting Data requestData |
| **Integration** | |
| Threat Diagram |   AD!{modifiyData} TD!{data} TargetDomain X AffectedDomain M ThreatAgent Human Deliberate B TA!{inject} AD!{forwardData} data AR: Injection knowledge inject |

### 4.2 Application Example

In 2017, the Open Web Application Security Project (OWASP) published a list of the ten most critical security risks for web applications [1]. We created a threat description with our template for all entries of that list. Tables 1 and 2 show the instance for the entry *Injection*. An injection as described by OWASP can be used to inject malicious code. Since there are different types of injections, we focus on the case of malicious database queries via the user input. There is no validation of the user input and the input is forwarded directly to the database. It is a deliberate threat performed by a human threat agent and can be performed via a wide area network, via a local network or locally. The domain to be affected is the machine, because the injection takes place on the software level where the user input is not validated. The target domain representing the database is a lexical domain. The complexity of the described threat is high, because an attacker needs deeper knowledge about the table structure of the database. Since the attacker acts as a user, only low privileges are required without any additional user interaction. The scope is changed because attacking the machine leads to an impact on the lexical domain. For all security goals, the impact is defined as high. The corresponding problem diagrams are given in Table 2. They have in common that an input by a user is possible. Using the provided input, a threat agent is able to inject malicious code. To integrate the threat into the initial set of problem diagrams, we define three join points. First, the lexical domain representing the database (target domain) needs to be mapped, as well as the machine domain (affected domain). Since the threat agent takes the role of user, the biddable domain of the the threat agent is a join point, too. The anti-requirement refers to the event of injection (phenomenon *inject*) and constrains the manipulated data at the target domain. The information that the threat agent might disclose by performing an injection is indicated by the constrained phenomena *knowledge*.

In the next section, we introduce a method to identify relevant threats based on elements of ProCOR (see Sect. 2.2) and the presented template. In the application example of the method, we make use of the instantiated template.

## 5 Threat Identification

In this section, we describe a semi-automatic procedure to identify relevant threats based on the previously described template. As input, we consider the domains in scope of the analysis that are derived with ProCOR (see Sect. 2.2). The procedure has five steps. Figure 2 shows an overview of the steps to be carried out. Steps that can be carried out automatically are shown in gray. Manual steps are shown in white. In the following, we first describe the steps of the procedure, and then we exemplify the procedure with a case study.

### 5.1 Procedure Steps

In the first step of ProCOR, we define the focus and scope of the security analysis based on problem diagrams. The scope can be described as shown in Table 3.

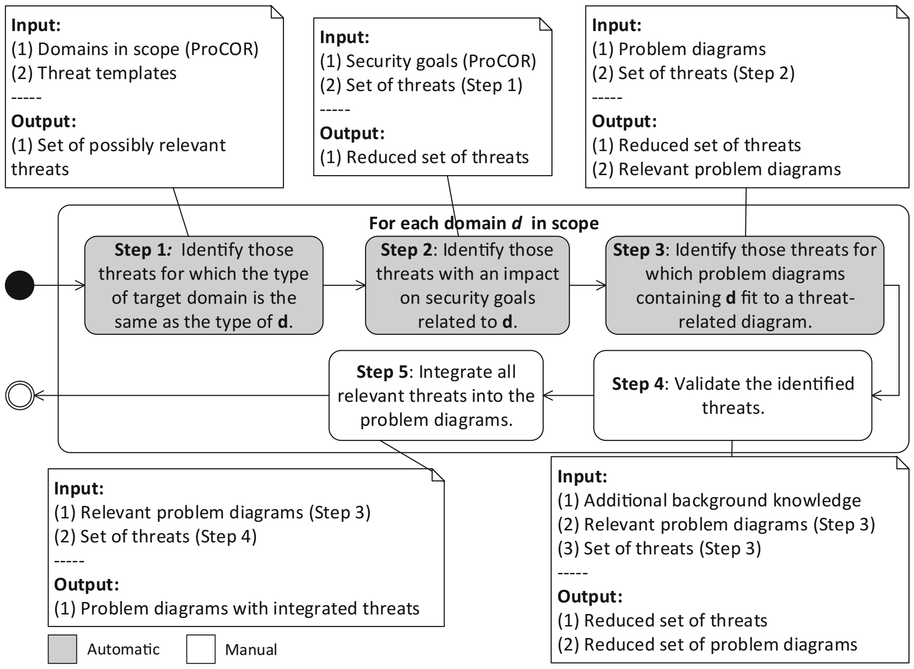

Input:
(1) Domains in scope (ProCOR)
(2) Threat templates
-----
Output:
(1) Set of possibly relevant threats
Input:
(1) Security goals (ProCOR)
(2) Set of threats (Step 1)
-----
Output:
(1) Reduced set of threats
Input:
(1) Problem diagrams
(2) Set of threats (Step 2)
-----
Output:
(1) Reduced set of threats
(2) Relevant problem diagrams
For each domain d in scope
Step 1: Identify those threats for which the type of target domain is the same as the type of d.
Step 2: Identify those threats with an impact on security goals related to d.
Step 3: Identify those threats for which problem diagrams containing d fit to a threat-related diagram.
Step 5: Integrate all relevant threats into the problem diagrams.
Step 4: Validate the identified threats.
Input:
(1) Relevant problem diagrams (Step 3)
(2) Set of threats (Step 4)
-----
Output:
(1) Problem diagrams with integrated threats
Input:
(1) Additional background knowledge
(2) Relevant problem diagrams (Step 3)
(3) Set of threats (Step 3)
-----
Output:
(1) Reduced set of threats
(2) Reduced set of problem diagrams
Automatic
Manual

**Fig. 2.** Procedure description

For each domain in scope, we document the piece of information that is in the focus of the analysis and that is available at this domain. For each piece of information, we also document the related security goals. Each step of our procedure has to be carried out for each domain **d** that is considered to be in scope.

*Step 1: Identify those threats for which the type of target domain is the same as the type of* ***d****.*

For this step, it is necessary to compare the type of **d** with the documented type of the target domain. The step can be realized automatically when using a model for the domains and when the template is encoded in a machine-readable way, for example XML.

*Step 2: Identify those threats with an impact on security goals related to* ***d****.*

All information in focus of the analysis that is available at **d** is documented along with the related security goals. We compare these security goals with the impact for confidentiality, integrity and availability documented in the template. Only those threats remain in the reduced set of threats which have such an impact. In this step, we do not distinguish between low and high impact for a security goal. The impact scales are used for calculating the risk level.

*Step 3: Identify those threats for which problem diagrams containing* **d** *fit to a threat-related diagram.*

As already mentioned, it does not suffice to consider the domain type and the security goal to decide whether a threat is relevant or not. In this step, we also take the functional requirements into account, which are expressed with problem diagrams. For each problem diagram in which **d** is contained, it has to be checked whether it is comparable with the diagrams given in the threat description. Comparable means that the domains, interfaces and dependencies mentioned in the threat description are also contained in the problem diagram under investigation. The relevant problem diagrams are collected to integrate the threats in the fifth step of the method.

*Step 4: Validate the identified threats.*

The previously mentioned steps can be performed automatically, based on the requirements model and threat templates. A manual validation of the identified threats is still necessary. For example, a threat might only be relevant for wireless connections. The type of connection is not given in the requirements model, and hence has to be evaluated by security engineers.

*Step 5: Integrate all relevant threats into the problem diagrams.*

The identified threats are integrated into the problem diagrams as described in Sect. 4 using the integration section of the underlying templates.

## 5.2 Application Example

Our case study is inspired by the OPEN meter project [12]. It describes a smart grid scenario in which an energy supplier is able to control the grid and to retrieve information such as the power consumption of customers. We performed our procedure for eight requirements, but in this paper we only show its application for the requirements *Setup*, *Measuring* and *Change Personal Data*. The communication hub is the piece of software to be developed and serves as the interface between customer's home and the energy supplier. The energy supplier has to perform an initial setup of the communication hub by entering the customer data and necessary tariff parameters, which are stored in the configuration. The corresponding problem diagram for this requirement is shown in Fig. 3. The communication hub can be used for automatic measuring and storing of the power consumption. The measuring component is called *SmartMeter* and measures the consumption in regular intervals to forward it to the communication hub. The measured data is stored as *MeterData*. Figure 4 shows the problem diagram for the requirement *Measuring*. A user is able to change his/her personal data using an interface provided by the communication hub. The personal data is stored in the configuration. The corresponding problem diagram is shown in Fig. 5. We limit our application example to the following assets: (1) Integrity of tariffParameters, (2) Availability of measuredData and (3) Integrity of measuredData.

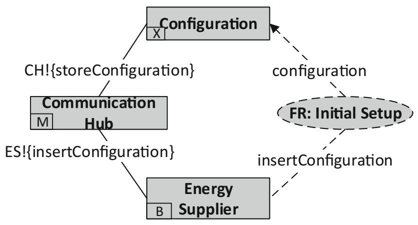


**Fig. 3.** Problem diagram *Setup*

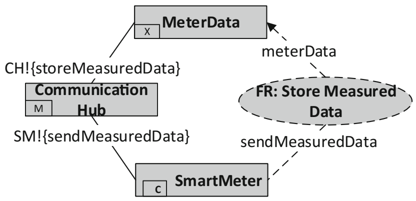


**Fig. 4.** Problem diagram *Measuring*

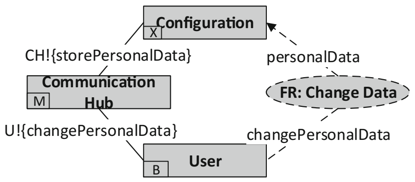


**Fig. 5.** Problem diagram *Change Personal Data*

Following the ProCOR method, we obtain the domains to be considered for the threat identification. The list of domains is shown in Table 3.

For all domains in scope (see Table 3), we perform the steps of the introduced procedure. We use the threat *Injection* described in Tables 1 and 2 as the input for the procedure.

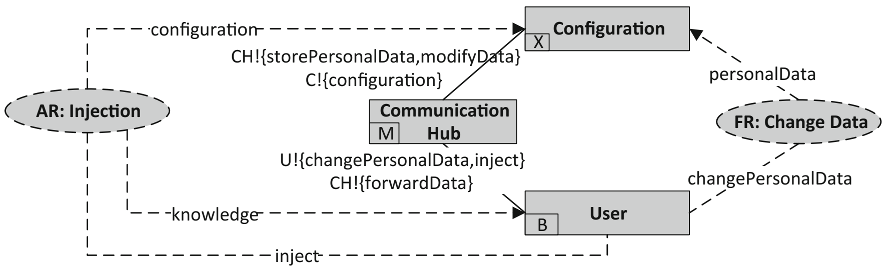


**Fig. 6.** Problem diagram with integration

**Configuration (X).** *Step 1:* The domain *Configuration* is a lexical domain. Comparing the domain type with the type of target domain documented for the threat, we consider the threat as relevant. *Step 2:* The security goal related to this domain is integrity. There is a high impact for integrity documented in the template. Hence, the threat remains relevant. *Step 3:* The domain is contained in the problem diagrams shown in Figs. 3 and 5. Both are comparable to the diagrams mentioned in the threat description. Hence, the threat is considered for the next step. *Step 4:* Considering the informal description of the threat, the problem diagram for *Setup* is not relevant. The input is only available for employees of the energy supplier. The threat agent acts as a user, but not as the energy supplier. The problem diagram *Change Personal Data* is relevant because the threat agent takes the role of a user. He/She can use the user interface to inject malicious code, which violates the security goal integrity. *Step 5:* The threat needs to be integrated into the problem diagram *Change Personal Data*. The join point for the machine is mapped to the communication hub, and the join point for the lexical domain is mapped to the configuration. The anti-requirement for injection is added to the problem diagram. All phenomena described in the threat diagram are added to the interfaces. The extended problem diagram is shown in Fig. 6.

**CommunicationHub (M).** *Step 1:* There is no threat mentioned with a machine as type of target domain. The other steps do not need to be carried out.

**EnergySupplier (B).** *Step 1:* There is no threat mentioned with a biddable domain as type of target domain. The other steps do not need to be carried out.

**Table 3.** Domains in scope

| Domain (Type) | Information | Security goal |
|---|---|---|
| Configuration (X) | tariffParameters | Integrity |
| CommunicationHub (M) | measuredData | Availability |
| | measuredData | Integrity |
| | tariffParameters | Integrity |
| EnergySupplier (B) | tariffParameters | Integrity |
| MeterData (X) | measuredData | Availability |
| | measuredData | Integrity |
| SmartMeter (C) | measuredData | Availability |
| | measuredData | Integrity |

**MeterData (X).** *Step 1:* The type of target domain of the threat *Injection* is the same as of *MeterData*. *Step 2:* The threat violates the security goals availability and integrity. Hence, the threat needs further consideration. *Step 3:* The domain *MeterData* is contained in the problem diagram shown in Fig. 4. The problem diagram is not comparable to the diagrams which are relevant for the threat *Injection*, because there is no user. Hence, the threat is not relevant for this domain, and the following steps do not need to be carried out.

**SmartMeter (C).** *Step 1:* There is no threat mentioned with a causal domain as type of target domain. The other steps do not need to be carried out.

As a result, we have determined that the threat *Injection* is relevant for the requirement *Change personal data.*

## 6 Related Work

There are numerous resources for threats, most of them focusing on attacks without using a common description format. There is also a lack of automatic identification methods for these resources. Moreover, some resources are restricted to a specific application context, such as web applications. In the following discussion, we use the terminology described in Sect. 3.

Lin et al. [11] propose abuse frames to analyze security requirements from an attacker's point of view. An anti-requirement is fulfilled when a threat initiated by an attacker is realized. Domains are considered as assets. The malicious machine of an abuse frame acts as the interface between attacker and asset domain. Comparable to problem frames, abuse frames are patterns to describe a typical attacker behavior. To use an abuse frame, it is composed with a base problem which is represented by a problem frame. Composing means to map domains from the base problem into the abuse frame. Based on the composed abuse frame, the attacker's behavior can then be further analyzed. In contrast, we consider some piece of information as an asset and our contribution is not restricted to attacks.

The Open Source Web Application Project [1] provides a list of the ten most severe security risks for web applications. The described risks have been used to evaluate our description template. A method to identify threats or to incorporate them with functional requirement is not given.

The Cloud Security Alliance (CSA) [13] provides a list of top threats for cloud computing. Threats are grouped in so called security concerns. To define such security concerns, the CSA makes use of articles about documented security incidents. The articles are determined using search engines, and the identified incidents are categorized to build a set of security concerns. The description format is informal, and the application context is restricted to cloud computing.

Uzunov and Fernandez [14] propose an extensible threat library. The library is based on a pattern for an abstract threat description. The definition of threat is similar to attack. The focus for this library is on distributed systems, but the pattern can be adapted for other needs. There is a strong relation to software architectures, whereas we focus on the relation to functional requirements.

Microsoft developed a method called STRIDE [15]. It is a popular security framework which is used to identify security threats. Using data flow diagrams for modeling the system and its behavior, threats are elicited based on existing threat categories: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of privilege. Each of these categories is a negative counterpart to a security goal. Using STRIDE, threats are identified based on data flow diagrams, whereas our approach is domain-based and considers functional requirements.

The *Bundesamt für Sicherheit in der Informationstechnik* [16] provides a catalogue describing incidents that might harm IT security. Incidents are not restricted to attacks. General aspects as fire harming the hardware are considered, too. The descriptions in this catalogues are informal and textual.

Lund et al. [17] propose a model driven risk management method, called CORAS. The terminology used in this work differs from the one we use. The identification of relevant threats is performed as a structured brainstorming where analysts meet experts with a specific domain knowledge. To document the results of the discussion, a specific graphical language has been developed. A common description format for threats is not used, and there is no relation to functional requirements.

Jürjens [18] describes an extension for UML to integrate security related information into UML diagrams. To model problem diagrams with UML, Côté et al. [19] provide a UML profile, called UML4PF to model problem frames. The consideration of UMLSec and UML4PF for our template might support the consistency between the requirement model and the security model.

Opdahl and Sindre [20] introduce misuse cases as an extension of use cases. A misuse case is related to the notion of an anti-requirement (cf. Sect. 4). We provide a more detailed view based on the software level and information flow.

## 7 Conclusion and Future Work

In the present paper, we propose a template to describe security threats for software-based systems. Our template follows the principle of security-by-design and is applicable during requirements engineering. A systematic risk identification requires a catalogue of threats. Using our template, security engineers can describe threats in a structured manner and a threat catalogue can be set up as needed. We do not restrict the approach to any specific domain.

By making technical decisions, new and more concrete threats might arise. To describe and identify those new threats, one can adapt our template for other steps of the software development lifecycle.

The consideration of domains in the template allows the linking to a problem-based model for functional requirements. We also showed how threats can be identified based on functional requirements. The integration into the requirements model ensures the consideration of relevant threats in the further steps of the risk management and software development process.

Currently, we consider all threats in isolation. It is obvious that there are dependencies between different threats. For example, disclosing the credentials of an administrator by performing an injection may lead to new relevant threats. We plan to elaborate a way to add these relations between threats to the template. The method to identify threats will then be carried out iteratively until no new threats can be identified.

We extended the CVSS specification with some additional information such as the type of threat and the types of affected and target domains. As future work, we aim to make use of the CVSS scoring system to estimate the severity of a threat. This estimation then supports the derivation of the risk level. We will also elaborate how the provided scales can be improved with regard to the level of details, e.g. more values to define the impact of the threat.

We believe that the template can be used to capture most threats, for example social engineering, as well. Those threats may not be related to functional requirements but can be described using the first part of the template (see Table 1). In that case, the second part of the template needs to be replaced by diagrams representing domain knowledge. For the future, we plan to also consider domain knowledge diagrams as an input for the threat identification method. These diagrams describe for example an information flow between two persons, which is not related to any functional requirement, but can be used to realize a threat.

Based on our template, we defined a semi-automatic method to identify relevant threats. A limitation of that method is that only threats described by such a template can be identified. After applying the method, an additional manual validation to identify additional threats should be considered. We aim to assist the security engineer as best as possible in performing that validation. A possible starting point are questionnaires which guide through the validation process.

As already mentioned in Sect. 3, controls are used to reduce a risk. We plan to provide a comparable template to describe such controls. The selection of relevant controls will be based on the identified threats and their integration into the requirements model.

## References

1. Open Web Application Security Project: OWASP Top 10 - The Ten Most Critical Web Application Security Risks (2017)
2. FIRST.org: Common Vulnerability Scoring System v3.0: Specification Document
3. Jackson, M.: Problem Frames. Analyzing and Structuring Software Development Problems. Addison-Wesley, Boston (2001)
4. Wirtz, R., Heisel, M., Meis, R., Omerovic, A., Stølen, K.: Problem-based elicitation of security requirements - the ProCOR method. In: Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering, vol. 1, pp. 26–38. ENASE, INSTICC, SciTePress (2018)
5. Faßbender, S., Heisel, M., Meis, R.: Aspect-oriented requirements engineering with problem frames. In: ICSOFT-PT 2014 - Proceedings of the 9th International Conference on Software Paradigm Trends. SciTePress (2014)
6. ISO: ISO 31000 Risk Management - Principles and Guidelines. International Organization for Standardization (2009)
7. Wirtz, R., Heisel, M., Borchert, A., Meis, R., Omerovic, A., Stølen, K.: Risk-based elicitation of security requirements according to the ISO 27005 standard. In: Evaluation of Novel Approaches to Software Engineering 13th International Conference, ENASE 2018. LNCS, Madeira, Portugal. Springer, Heidelberg (2018, submitted for publication)
8. International Organization for Standardization: ISO 27005:2011 Information technology - Security techniques - Information security risk management. Standard (2011)
9. Common Criteria: Common Criteria for Information Technology Security Evaluation v3.1. Release 5. Standard (2017)

10. FIRST.org: Common Vulnerability Scoring System v3.0: User Guide
11. Lin, L., Nuseibeh, B., Ince, D.C., Jackson, M., Moffett, J.D.: Analysing security threats and vulnerabilities using abuse frames (2003)
12. OPEN meter Consortium: Report on the identification and specification of functional, technical, economical and general requirements of advanced multi-metering infrastructure, including security requirements (2009)
13. Cloud Security Alliance: The treacherous 12 - cloud computing top threats in 2016
14. Uzunov, A., Fernandez, E.: An extensible pattern-based library and taxonomy of security threats for distributed systems. Comput. Stand. Interfaces **36**, 734–747 (2014)
15. Shostack, A.: Threat Modeling: Designing for Security. Wiley, Hoboken (2014)
16. BSI Germany: IT-Grundschutz-Katalog (2018)
17. Lund, M.S., Solhaug, B., Stølen, K.: Model-Driven Risk Analysis. The CORAS Approach. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12323-8
18. Jürjens, J.: Model-based security engineering with UMLsec. In: Serenity Day: Establishing IT Security as a Full Engineering Discipline, Brussels (2009)
19. Côté, I., Hatebur, D., Heisel, M., Schmidt, H.: UML4PF - a tool for problem-oriented requirements analysis. In: Proceedings of the International Conference on Requirements Engineering (RE). IEEE Computer Society (2011)
20. Sindre, G., Opdahl, A.L.: Eliciting security requirements with misuse cases. Requir. Eng. **10**(1), 34–44 (2005)