

# Risk Identification: From Requirements to Threat Models

Roman Wirtz and Maritta Heisel  
University of Duisburg-Essen, Germany

Keywords: Risk Management, Security, Risk Identification, Threats, Requirements Engineering.

Abstract: Security is a key factor for providing high-quality software. In the last few years, a significant number of security incidents has been reported. Considering scenarios that may lead to such incidents right from the beginning of software development, i.e. during requirements engineering, reduces the likelihood of such incidents significantly. Furthermore, the early consideration of security reduces development effort since identified scenarios do not need to be fixed in later stages of the development lifecycle. Currently, the identification of possible incident scenarios requires high expertise from security engineers and is often performed in brainstorming sessions. Those sessions often lack a systematic process which can lead to overlooking relevant aspects. Our aim is to bring together security engineers and requirements engineers. In this paper, we propose a systematic, tool-based and model-based method to identify incident scenarios based on functional requirements by following the principle of security-by-design. Our method consists of two parts: First, we enhance the initial requirements model with necessary domain knowledge, and second we systematically collect relevant scenarios and further refine them. For all steps, we provide validation conditions to detect errors as early as possible when carrying out the method. The final outcome of our method is a CORAS threat model that contains the identified scenarios in relation with the requirements model.

## 1 INTRODUCTION

Security is a key factor for providing high-quality software. In the last few years, a significant number of security incidents have been reported. Such incidents can lead to strong consequences for software providers not only financially, but also in terms of reputation loss. By following the principle of security-by-design and considering security right from the beginning of a software development lifecycle, the consequences for software providers can be reduced significantly. Besides, the early consideration of security reduces development effort since identified security aspects do not need to be fixed in the later stages of the development lifecycle.

A central aspect for providing secure software is the identification of scenarios that may lead to harm for a stakeholder's asset, i.e. a piece of value for that stakeholder. The identification is an essential part of risk management activities and requires high expertise from security engineers. Risk management deals with the identification and treatment of incidents in an effective manner by prioritizing them according to their risk level. The level can be defined by an incident's likelihood and its consequence for an asset (International Organization for Standardization, 2018).

Currently, development teams often perform brainstorming sessions for that purpose, in which software engineers and security engineers come together and share their specific expertise. Those sessions often lack a systematic process which can lead to overlooking relevant aspects.

We aim to bring together security engineers and requirements engineers by providing systematic guidance in identifying relevant incident scenarios as early as possible, i.e. during requirements engineering. In previous work (Wirtz et al., 2018), we described a risk management process which combines problem frames with CORAS. In this method, the identification of threats relies on external knowledge and lacks of a systemization. To address this issue, we further proposed a template to specify threats based on the *Common Vulnerability Scoring System* (Wirtz and Heisel, 2018; FIRST.org, 2015). The template uses problem diagrams to relate specified threats to functional requirements. The method we introduced to exemplify the usage of the template lacks of a process to identify relations between different threats.

In the present paper, we present a method that allows to systematically derive CORAS threat diagrams (Lund et al., 2011) from functional requirements of software to be developed. With our method, we fo-

cus on information security, and therefore we define a piece of information as an asset to be protected with regard to confidentiality, integrity, or availability. A CORAS threat diagram describes scenarios that might lead to harm for an asset concerning one of these security properties. Our method follows a problem-oriented approach. Therefore, we consider a problem frames model as required input (Jackson, 2000). First, we define security goals and elicit additional domain knowledge, which we both document in a model. We proceed with systematically identifying relevant scenarios based on the requirements model. The documented security goals help to focus on the relevant aspects of the model. For the documentation, we use a security model that is based on the CORAS language (Lund et al., 2011). The last step of our method is the refinement of the model, for which we inspect the identified scenarios in more detail.

To support the application of our method, we provide a graphical tool. It allows to document the results of the method in a model and provides different views on the model instances. The model-based approach ensures consistency and traceability. The tool guides through the different steps of our method and supports their execution, thus limiting the manual effort to apply the method. Furthermore, we define validation conditions for each step which help to detect errors in the application of the method as early as possible.

The remainder of the paper is structured as follows: In Section 2, we briefly introduce the underlying concepts of this paper, i.e. problem frames and CORAS, followed by a description of the underlying models in Section 3. Section 4 contains our identification method which we exemplify in Section 5. We discuss related work in Section 6 and conclude our paper with an outlook on future research directions in Section 7.

## 2 BACKGROUND

For our method, we mainly consider two fundamental concepts which we introduce in the following.

### 2.1 Problem Frames & Domain Knowledge

To model functional requirements, we make use of the problem frames approach as introduced by Michael Jackson (Jackson, 2000). For modeling requirements, we make use of problem diagrams which consist of domains, phenomena, and interfaces. In previous work, we proposed a notation based on the Google

Material Design<sup>1</sup> which provides a user-friendly way to illustrate the diagrams (Wirtz and Heisel, 2019).

Machine domains ( $\square$ ) represent the piece of software to be developed.

Problem domains represent entities of the real world. There are different types: biddable domains with an unpredictable behavior, e.g. persons ( $\odot$ ), causal domains ( $\otimes$ ) with a predictable behavior, e.g. technical equipment, and lexical domains ( $\square$ ) for data representation. A domain can take the role of a connection domain ( $\boxplus$ ), connecting two other domains, e.g. user interfaces.

Interfaces between domains consist of phenomena. There are symbolic phenomena, representing some kind of information or a state, and causal phenomena, representing events, actions, and commands. 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 one domain indicated by  $X!\{\dots\}$ , where  $X$  stands for an abbreviation of the name of the controlling domain.

A problem diagram contains a statement in form of a functional requirement (represented by the symbol  $\boxminus$ ) describing a specific functionality to be developed. A requirement is an optative statement which describes how the environment should behave when the software is installed.

Some phenomena are *referred to* by a requirement (dashed line to controlling domain), and at least one phenomenon is *constrained* by a requirement (dashed line with arrowhead and italics). 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.

In Figure 1, we show a small example describing a functional requirement for updating some information. A *Person*  $\odot$  provides information to *Software*  $\square$  to be updated. We make use of a lexical domain *Information*  $\square$  to illustrate a database. The functional requirement *Update*  $\boxminus$  refers to the phenomenon *updateInformation* and constrains the phenomenon *information*.

In addition to problem diagrams, we make use of domain knowledge diagrams which have a similar syntax as problem diagrams. Such diagrams do not contain any requirements, but indicative statements

<sup>1</sup>Google Material - <https://material.io> (last access: September 30, 2019)

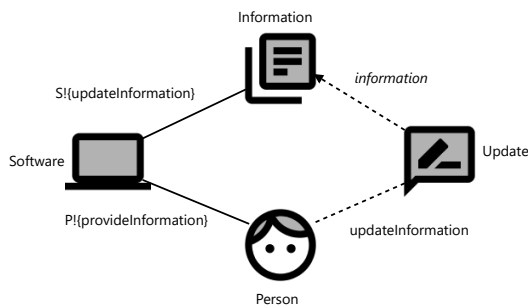


Figure 1: Example for Problem Diagram.

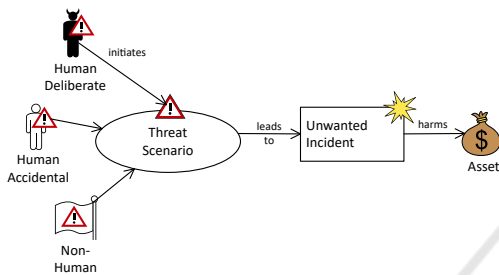


Figure 2: CORAS Threat Diagram.

which describe the environment. We distinguish two types of those statements, namely *facts* (□) and *assumptions* (▣) (van Lamswerde, 2009). As facts, we consider fixed properties of the environment irrespective of how the machine is built. Assumptions are conditions we have to rely on to be able to satisfy the requirements. Similar to requirements, those statements can *refer to* or *constrain* phenomena describing the expected behavior of domains according to the domain knowledge.

## 2.2 CORAS

CORAS (Lund et al., 2011) is a model-based method for risk management. It consists of a step-wise process and different kinds of diagrams. The method follows the ISO 31000 risk-management standard (International Organization for Standardization, 2018). Each step provides guidelines for the interaction with the customer on whose behalf the risk management activities are carried out. The results are documented in a model using the CORAS language. The method starts with the establishment of the context and ends up with the suggestion of treatments to address the risk.

In our method, we use the CORAS language to document incident scenarios, which may lead to a harm for an asset. The symbols we make use of are shown in Figure 2. *Direct Assets* are items of value. There are *Human-threats deliberate*, e.g. a

network attacker, as well as *Human-threats accidental*, e.g. an employee pressing a wrong button accidentally. To describe technical issues there are *Non-human threats*, e.g. malfunction of software. A threat initiates a *Threat scenario*. A threat scenario describes a state, which may possibly lead to an unwanted incident, and an *Unwanted incident* describes the action that actually harms an asset.

## 3 UNDERLYING MODELS

To document the results of our method, we follow a model-based approach. We distinguish between a problem frames model and a CORAS threat model. For both models, we provide different views that show only parts of the model to help users of our method focusing on relevant aspects, thus ensuring scalability.

**Problem Frames Model.** The problem frames model contains domains, phenomena, and interfaces to document and analyze statements, i.e. requirements and domain knowledge (cf. Section 2.1). With regard to our method, there are three different views on the model: (i) Statement diagrams to show selected requirements and domain knowledge; (ii) an information flow graph (IFG) to analyze the information flow between domains; and (iii) a statement graph to show the relations between domains and statements.

**Security Model.** The security model holds a reference to the problem frames model to ensure traceability and consistency. In the security model, we document security goals for software under development and identified security incidents using the CORAS approach. For this reason, we provide two views on the model: (i) an overview of the defined security goals; and (ii) CORAS threat diagrams describing the derived scenarios.

The tool, which we provide to support the application of our method makes use of these models, helps to maintain them, and helps to create graphical instances of the views.

## 4 METHOD

Our method allows to systematically derive security incidents from functional requirements. It consists of two parts with different sub-steps, for which we provide an overview in Figure 3. In the following, we describe the steps in detail and state examples from a growing set of validation conditions. Note, that some conditions can be checked automatically using our

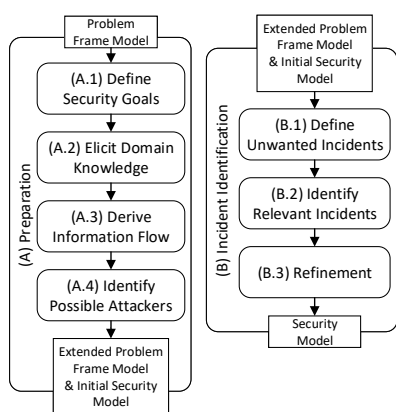


Figure 3: Method overview.

tool. The ones to be checked manually help security analysts in focusing on specific parts while reviewing the documented results.

### 4.1 (A) Preparation

As input for our method, we require a problem frame model describing the functional requirements of software to be developed. Before identifying threats, it is necessary to elicit domain knowledge and to make that knowledge explicit. Domain knowledge describes specific characteristics of the environment, which complements the requirements model. With regard to security, domain knowledge is an important aspect since it may include additional threats, e.g. social engineering. Within the preparation part of the method, we define four sub-activities for which we document the results in the model, thus providing an extended problem frame model as the outcome.

#### Step A.1: Define Security Goals

An asset is something of value for a stakeholder. Since our method concentrates on information security, we consider a piece of information which is of value for someone as an asset. A security goal denotes that an asset shall be protected with regard to confidentiality, integrity or availability.

As input for the step, we consider the initial problem frame model describing the functional requirements. In a problem frame model, information is represented by symbolic phenomena. Therefore, we take the list of all symbolic phenomena as input for which domain experts, e.g. the software provider, have to define necessary security goals which are documented in the security model (cf. Section 3). Each security goal holds a reference to the corresponding phenomenon in the problem frame model.

There are two validation conditions for the first

step that concern the completeness of security goal definition.

VC1. All symbolic phenomena of the initial requirements model have been considered as possibly being an asset.

VC2. All desired security goals have been specified and have been documented in the security model.

#### Step A.2: Elicit Domain Knowledge

The second sub-activity deals with eliciting additional domain knowledge. The initial problem frame model only covers functional requirements, and therefore only captures those domains and interfaces that belong to those requirements. For identifying threats, it is essential to elicit domain knowledge. We consider the identification of additional stakeholders, as well as additional information flows. To document the results of the elicitation along with functional requirements, we make use of domain knowledge diagrams as described in 2.

Meis proposes questionnaires to elicit domain knowledge based on problem frame models in the context of privacy (Meis, 2013). There are different questionnaires per domain type, i.e. one for causal and lexical domains and one for biddable domains. We adapt those questionnaires to identify stakeholders and information flows, which we show in Table 1.

We define three validation conditions for the present step which concerns the completeness of domain knowledge elicitation. Another important aspect is that we only capture the desired characteristics of the environment at this stage, not for example attackers.

VC3. All domains have been inspected using the provided questionnaires.

VC4. The identified domain knowledge only captures the expected and desired characteristics of the environment with regard to the software to be developed.

Table 1: Questionnaire for indirect stakeholders.

No	Question
<i>Biddable Domains</i>	
1	Does the domain share any information with other biddable domains?
2	Does the domain receive any information from other biddable domains?
<i>Causal &amp; Lexical Domains</i>	
3	Are there stakeholders in the environment that have direct access to the domain?
4	Are there other systems that contain the domain? What stakeholders have access to the domain?

VC5. The identified domain knowledge has been documented in the problem frame model.

### Step A.3: Derive Information Flow

To identify threats for assets, it is necessary to further analyze the intended information flow for the system and to identify those domains at which an asset is available. On the one hand, with regard to confidentiality, it is necessary to not disclose the asset to other domains than intended. On the other hand, integrity and availability of assets shall be preserved at the identified domains.

We make use of an information flow graph to derive the intended information flow (Wirtz et al., 2018). Such a graph is directed and depicts which information represented as a phenomenon is available at which domain. The graph can be created by inspecting the interfaces between the domains of a problem frame model. When both domains share an asset-related phenomenon, it means that the information is available at both domains.

We limit the information flow graph to those phenomena that are an asset, i.e. to those phenomena for which a security goal has been defined. We do not consider attackers we identify in the following step for the information flow graph since assets shall not be available for them.

For the third step, we define the following validation conditions:

VC6. All domains and interfaces of the problem frame model have been considered for the derivation of the information flow graph.

VC7. The graph only depicts domains at which an asset is available.

### Step A.4: Identify Possible Attackers

In *Step A.2*, we focused on intended and authorized stakeholders and information flows in the environment. In the present step, we identify possible attackers by inspecting each domain of the problem frame model with regard to its vulnerability. Again, we make use of questionnaires with respect to the type of domain under investigation, which we show in Table 2. For a biddable domain, we investigate if the domain is vulnerable for social engineering attacks or if it may act as an attacker itself, e.g. as a malicious insider. For causal and lexical domains, we elicit attackers that may attack that domain.

Using domain knowledge diagrams, we document the attackers in the problem frame model. Such a diagram consists of a statement, attacker domain, vulnerable domain, and the interface between both domains.

Note, that the identification of possible attackers is an approximation since we neither define the objectives of an attacker nor the conditions under which the attack may succeed. The detailed analysis of threats follows in the second phase of the method.

There are three validation conditions for this step with regard to completeness and documentation of identified attackers.

VC8. Each domain has been considered for being possibly vulnerable.

VC9. All attackers have been documented independently of whether an attack may be successful or not.

VC10. The domain knowledge diagrams describe the identified attackers in a meaningful manner.

## 4.2 (B) Incident Identification

The extended problem frame model and initial security model with security goals serve as the input for the threat identification. For identifying threats, we consider the system as a whole, which means that we do not consider each requirement or domain knowledge in isolation, but also consider relations between them. Considering a combination of statements ensures to capture a complete path of actions throughout different domains that are necessary to successfully realize a threat. As mentioned in Section 2, we make use of the CORAS notation (Lund et al., 2011) to document threats. The threat identification consists of three sub-activities which we describe in the following. We document the results in the form of CORAS threat diagrams in the security model.

### Step B.1: Define Unwanted Incidents

We create a CORAS threat diagram backwards, starting from an asset, for which we first define unwanted incidents. An unwanted incident denotes the action in which an asset is actually harmed with regard to its defined security goal. For each security property, we consider a specific type of unwanted incident, and for each type, we provide a textual pattern (TP), where “< ... >” denotes a variable part of the pattern.

Table 2: Questionnaire for attackers.

No	Question
<i>Biddable Domains</i>	
1	Is the domain vulnerable for social engineering attacks?
2	Is it possible that the domain may act as an attacker, e.g. as a malicious insider?
<i>Causal &amp; Lexical Domains</i>	
3	Could the domain be attacked, by whom and how?

We define unwanted incidents for each security goal in the following way:

**Confidentiality.** In the context of confidentiality, an unwanted incident denotes an undesired disclosure of an asset to unauthorized third parties represented as a biddable domain in the model. We identify unauthorized third parties with the help of the information flow graph which shows at which domains an asset is available. As relevant third parties, we consider those biddable domains at which an asset is *not* available, i.e. which are not contained in the information flow graph with regard to this asset. Security analysts now decide whether the disclosure to this domain shall be considered for further analysis. In case of consideration, we create an unwanted incident with the following textual pattern, where *third party* describes the domain to which the asset is disclosed:

**TP:**  $\langle \textit{asset} \rangle$  is disclosed to  $\langle \textit{third party} \rangle$ .

**Integrity.** An unwanted incident with regard to integrity is the unauthorized alteration of an asset at a specific domain. The information flow graph depicts all domains at which an asset is available. For each identified domain, security analysts have to decide whether the harm of integrity at this domain shall be considered for further analysis. Using the following textual pattern, we document the corresponding unwanted incident, where *domain* stands for the domain at which the integrity may be harmed.

**TP:**  $\langle \textit{asset} \rangle$ 's integrity is violated at  $\langle \textit{domain} \rangle$ .

**Availability.** For availability, an unwanted incident denotes that an asset is not available for authorized entities. As authorized entities, we consider all domains of the information flow graph at which the asset is available. Again, security analysts have to decide whether the unavailability at this domain shall be further considered. The following textual pattern describes the resulting unwanted incident:

**TP:**  $\langle \textit{asset} \rangle$  is unavailable at  $\langle \textit{domain} \rangle$ .

We define the following three validation conditions for the present step:

VC11. All security goals have been considered for the definition of unwanted incidents.

VC12. For confidentiality, all biddable domains at which the asset shall not be available have been considered for identifying relevant unwanted incidents.

VC13. For integrity and availability, all domains of the information flow graph have been considered at which the corresponding asset is available.

### Step B.2: Identify Relevant Incidents

After we defined unwanted incidents that shall be considered, we identify threat scenarios and threats that may lead to such unwanted incidents. To do so, we do not only focus on specific statements but consider the system as a whole, i.e. all requirements and domain knowledge. We identify sequences of statements that may lead to an unwanted incident. A statement that refers to a domain and constrains another indicates a possible information flow from the referred domain to the constrained one. The statement graph of the requirements model provides an overview of all statements and their references to domains. A path in the graph (sequence of statements) denotes an information flow via several domains. In the following, we identify relevant paths depending on the type of security property of an unwanted incident and translate those sequences of statements systematically into a CORAS threat diagram.

" $\langle \dots \rangle$ " represents a sequence of statements.

**Confidentiality.** With regard to confidentiality, we collect all non-cyclic paths  $\langle s_1, \dots, s_n \rangle$  for which the following conditions hold:

- $s_1$  constrains the domain that is given in the unwanted incident, i.e. the stakeholder to which the asset shall not be disclosed.
- $s_n$  refers to a domain at which the asset is available.
- For all  $1 \leq i < n$ :  $s_i$  refers to a domain that  $s_{i+1}$  constrains.

Such a path denotes a sequence of statements that may lead to a disclosure of the asset to the unauthorized stakeholder. The path can be translated into a CORAS threat diagram in the following way:

- Each statement  $s_i$  is translated to a threat scenario  $ts_i$ .
- $ts_1$  leads to the unwanted incident.
- For all  $1 \leq i < n$ :  $ts_{i+1}$  leads to  $ts_i$

Translating a statement into a threat scenario means to identify a situation with regard to the statement that may lead to information flow from the referred domain to the constrained domain, assuming that the asset is available at the referred domain. The resulting path denotes a sequence of threat scenarios that may lead to an information flow of the asset from a domain at which it is available to the unauthorized stakeholder. For each threat scenario, we also identify the domain that initiates it. For example, an attacker is a human-threat deliberate who realizes a social engineering attack. We add the threat to the threat diagram, as well. In case that a

statement of a path cannot be translated into a threat scenario, e.g. because involved domains are considered as secure, we do not add the path to the model.

**Integrity.** In the following, we identify sequences of statements that could lead to harm of integrity for a given asset. We consider paths of statements  $\langle (d_1, s_1), \dots, (d_n, s_n) \rangle$  for which the following conditions hold:

- $d_1$  is a domain at which the integrity shall be preserved
- For all  $1 \leq i \leq n$ :  $s_i$  constrains  $d_i$ .
- For all  $1 \leq i < n$ :  $s_i$  refers to  $d_{i+1}$ .
- $\forall i, j : 1..n | i \neq j \bullet d_i \neq d_j$

We do not consider a specific domain  $d_n$  as an attacker that harms the integrity of an asset at  $d_1$ . Instead, we consider all non-cyclic paths of statements that may lead to harm of integrity. Those paths can be translated into a CORAS threat diagram in the following way:

- Each statement  $s_i$  is translated to a threat scenario  $ts_i$
- $ts_1$  leads to the unwanted incident for  $d_1$
- For all  $1 \leq i < n$ :  $ts_{i+1}$  leads to  $ts_i$

For each threat scenario, we also document the corresponding threat that initiates the threat scenario. Each path of threat scenarios denotes a sequence that may lead to harm of integrity for an asset at a given domain.

**Availability.** The creation of threat scenarios for unwanted incidents with regard to availability is similar to the consideration of integrity. We identify paths in the same way but consider threat scenarios that may lead to unavailability of the asset at a given domain.

For the present step, we identify the following validation conditions:

- VC14. To identify statements paths, all statements contained in the model have been considered.
- VC15. Each identified path has been translated to a corresponding path in the threat diagram. If not possible, justifications have been given.
- VC16. Only non-cyclic paths have been considered.
- VC17. The domains that realize a threat scenario have been documented as threats.

### Step B.3: Refinement

To create the CORAS threat diagram in the previous step, we only consider one threat scenario per statement. To further refine the threat scenario, we inspect

the corresponding statement in detail by analyzing its problem diagram or domain knowledge diagram, respectively. For each domain or interface of the diagram, we identify threat scenarios that may cause the initial scenario and add them to the model. Each of them may lead to the initial one, thus pointing to it.

The step of refinement is optional. A challenge for security analysts is to find the right abstraction level. They should only refine threat scenarios where necessary to not provide an overhead of information.

For the last step of our method, we define the following validation conditions:

VC18. Only those domains have been considered to refine a threat scenario that are related to the corresponding statement, i.e. contained in its problem diagram or domain knowledge diagram.

VC19. The identified threat scenarios point to the initial scenario, thus refining it.

VC20. Refinements have only been made where necessary.

The resulting threat diagram can now be used as a starting point to evaluate risks and to later select appropriate controls that treat the identified threats.

## 5 EXAMPLE

For reasons of simplicity, we consider a small example representing a typical software for a company. We also applied our method to more complex examples, e.g., a smart home scenario, to validate our approach and the results.

All outcomes we show in the following can be created with our tool. The items are contained in the underlying models, and the diagrams provide different views (cf. Section 3).



### 5.1 Initial Input

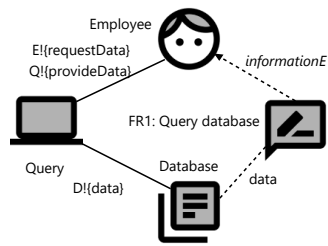
In the example, there are employees and a boss. For the software to be developed, we consider the following functional requirements (FR):

$FR_1$ . An employee can request data from a database.

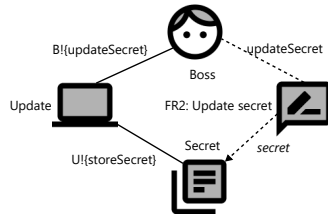
$FR_2$ . A boss maintains a secret which he/she can store and update in a database.

$FR_3$ . Data in the database shall be signed with the secret provided by the boss.

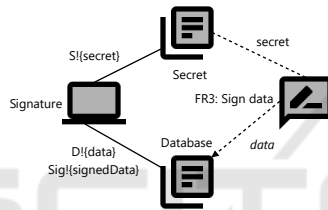
In Figure 4, we show the corresponding problem diagrams for the functional requirements. The requirement  $FR_1$ : *Query database*  refers *data* at the *Database* , where the data to be retrieved is stored.



(a)  $FR_1$ : Query Database



(b)  $FR_2$ : Update secret



(c)  $FR_3$ : Sign data

Figure 4: Example: Problem diagrams.

The requirement constrains the phenomenon *informationE* of the *Employee* since to satisfy the requirement the employee has to receive the requested data. The machine *Query* describes the piece of software to be built to satisfy the requirement.

For the requirement  $FR_2$ : *Update secret*, we consider three domains: *Boss*, *Update*, and *Secret* which represents the specific database where the secret is stored. The requirement refers to the event *updateSecret* of the boss and constrains the secret.

The requirement  $FR_3$ : *Sign data* refers to the secret and constrains the data to be signed at the database. The machine *Signature* is responsible for signing the data.

## 5.2 Preparation

For the preparation, we consider the previously mentioned functional requirements as input.

**Define Security Goals.** We concentrate on the following security goals which are documented in the security model: (1) The integrity of the *secret* shall be preserved, and (2) the confidentiality of the *secret*

shall be preserved. We do not show how to identify threats with regard to availability since the procedure is similar to integrity.

**Elicit Domain Knowledge.** Using our proposed questionnaires, we identified two assumptions (A) as domain knowledge to be considered:

- A<sub>1</sub>. An employee provides information to the boss, e.g. about his progress in projects.
- A<sub>2</sub>. A boss provides some information to the employee, e.g. tasks to be performed.

Figure 5 shows the domain knowledge diagram for the stated assumptions which we document in the problem frame model. For example, the assumption *A1: Talk to boss* refers to the phenomenon *talkToBoss* and constrains the phenomenon *informationB*, which means that there is a communication from the employee to the boss. The assumption *A2: Receive information from boss* can be expressed in the same way.

**Derive Information Flow.** Using the information flow graph (Wirtz et al., 2018), we identify domains at which the asset *secret* is available. There are four domains we have to consider: (1) *Boss*, (2) *Update*, (3) *Secret*, and (4) *Signature*.

**Identify Possible Attackers.** Using our questionnaire, we identified an employee as a vulnerable domain for social engineering. This leads to the following additional assumptions to be considered for which we show the corresponding domain knowledge diagrams in Figure 6:

- A<sub>3</sub>. An attacker may influence an employee’s behavior, e.g. by performing a social engineering attack.
- A<sub>4</sub>. An attacker may receive secret information from an employee, e.g. by performing a social engineering attack.

In the diagram, we consider a new biddable domain *Attacker* which represents the identified attacker.

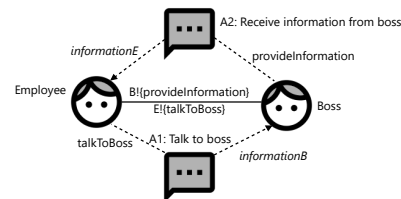


Figure 5: Example: Domain knowledge.



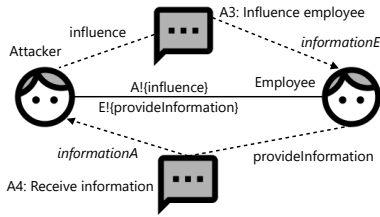


Figure 6: Example: Attackers.

### 5.3 Incident Identification

Using the extended problem frame model, we proceed with identifying threats.

**Define Unwanted Incidents.** With regard to the asset *secret*, we defined security goals for integrity and confidentiality. Those unwanted incidents are documented in the security model and serve as the starting point for creating a threat diagram. From the information flow graph we know that the asset shall not be available for *Employee* and *Attacker*. In the following, we will focus on confidentiality with regard to the attacker and define the following unwanted incident (UI):

$UI_1$ . *secret* is disclosed to *Attacker*.

For defining unwanted incidents with regard to integrity, we consider those domains at which the asset is available. We focus on the domain *Secret* and define the following unwanted incident:

$UI_2$ . *secret*'s integrity is violated at *Secret*.

We identify relevant scenarios for both unwanted incidents in the following.

**Identify Relevant Incidents.** To identify relevant statement paths, we focus on the statement graph, which is another view on the model. It only contains the statements and references to the domains. For our example, we show the graph in Figure 7. The notation of the graph is similar to problem diagrams:  $FR_2$ : *Update secret* refers to *Boss* and constrains *Secret*, thus indicating a possible information flow from boss to secret. In the following, we will use the graph to identify relevant threat scenarios. By systematically traversing the graph as described in Section 4, we identify relevant paths.

**Confidentiality.** The asset shall not be disclosed to *Attacker*. Therefore, we start at this domain and identify paths to domains at which the asset is available. For our example, this leads to the following paths (P):

$P_1$ :  $\langle A_4, FR_1, FR_3 \rangle$  (asset available at *Secret*)

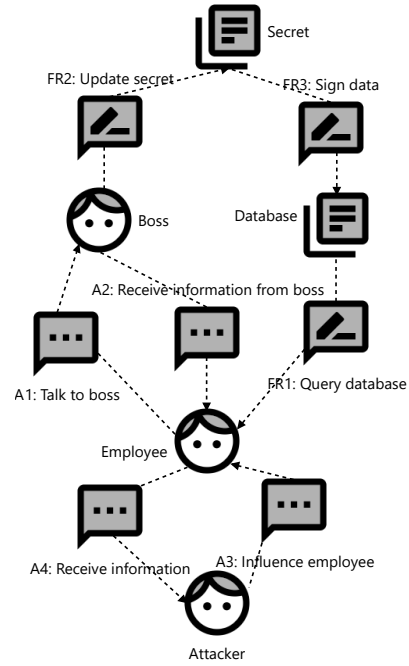


Figure 7: Example: Statement Graph.

$P_2$ :  $\langle A_4, A_2 \rangle$  (asset available at *Boss*)

We translate both paths into a CORAS threat diagram, which we show in Figure 8. It consists of the unwanted incident that harms the confidentiality of the asset *secret*. For each statement that is contained in a path, we add a threat scenario to the diagram. Both paths contain the statement  $A_4$  which can be translated into a threat scenario concerning a social engineering attack. The attacker is a deliberate human threat and the employee is an accidental human threat.  $P_1$  contains two additional statements. With regard to  $FR_1$  the machine *Query* may have a malfunction and is, therefore, a non-human threat. The malfunction may lead to an undesired information flow from the database to the employee. The same holds for the statement  $FR_3$  where *Signature* is a non-human threat that may lead to an unintended flow of the secret to the database.  $P_2$  contains the statement  $A_2$  where the boss provides some information to an employee. As an accidental human threat, the boss may provide the secret to the employee. Both paths show how the unwanted incident may be realized by following the path of threat scenarios.

**Integrity.** The integrity of the asset shall be preserved at the domain *Secret*, which leads to the following paths starting at the domain:

$P_3$ :  $\langle (Secret, FR_2), (Boss, A_1), (Employee, A_2) \rangle$

$P_4$ :  $\langle (Secret, FR_2), (Boss, A_1), (Employee, A_3), (Attacker, A_4) \rangle$

$P_5: \langle (Secret, FR_2), (Boss, A_1), (Employee, FR_1), (Database, FR_3) \rangle$

In Figure 9, we show the CORAS threat diagram including the paths  $P_3$  and  $P_4$ . The path  $P_5$  can be translated in the same way. For example, considering path  $P_4$ , the threat scenario derived from  $FR_2$  leads to the unwanted incident. It describes that an update performed by the boss may harm the integrity due to a malfunction of software, thus leading to the unwanted incident. The machine *Update* is a non-human threat and the boss an accidental human threat. Due to assumption  $A_1$ , an employee may provide incorrect information to the boss which may influence the update of the secret. A social engineering attack on an employee ( $A_3$ ) may yield to this provision of wrong information, and an attacker may use collected information from the employee to perform the attack ( $A_4$ ).

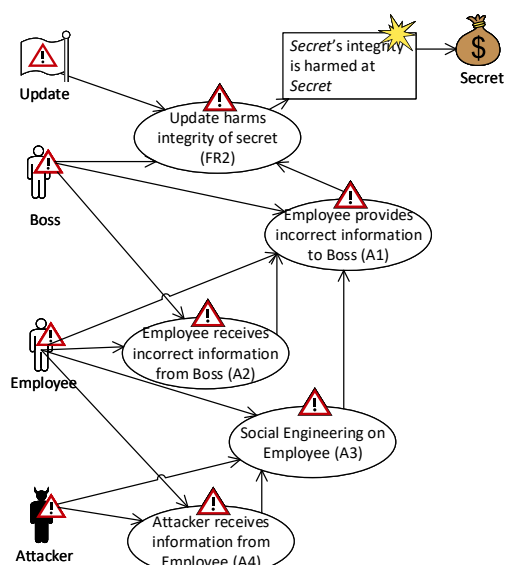





Figure 9: Example: Threats to integrity.

**Refinement.** Traversing the requirements model in the proposed way leads to a detailed analysis of dependencies between threat scenarios. We now show how such a threat scenario can be further refined. As an example, we consider the threat scenario *Update harms the integrity of secret* which belongs to the requirement  $FR_2$ : *A boss maintains a secret which he/she can store and update in a database*. In Figure 4 b) we show the corresponding problem diagram. To refine the threat scenario, we consider the domains and interfaces of the diagram. There are three domains to be considered: (1) *Boss* , (2) *Update* , and (3) *Secret* .

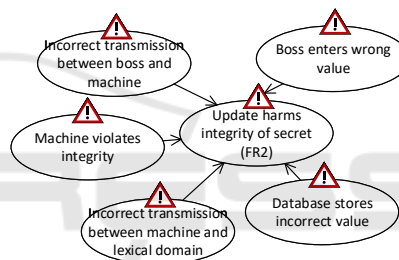


Figure 10: Example: Refinement.

In Figure 10, we show the resulting threat scenarios. For each domain and for each interface between domains, there is one threat scenario. For example,

the boss may enter the wrong value for the new secret.

The results of the threat identification can now be further analyzed, e.g. to identify appropriate controls to limit the harm for assets.



Figure 8: Example: Threats to confidentiality.

## 6 RELATED WORK

There is a recent case study on threat modeling that revealed the effectiveness and efficiency of threat modeling in large scale applications (Stevens et al., 2018). The participants reported that threat modeling supports the communication in teams and the development of mitigation strategies. Since we provide a tool and graphical representations of the model, we improve the scalability for larger applications, and the diagrams can be used as a starting point in discussions.

Lund et al. propose the CORAS risk management process from which we adapted our notation (Lund et al., 2011). The authors describe the process of

identifying threats as an interactive brainstorming session. By considering our method, the brainstorming sessions may be structured in a systematic way.

Abuse frames describe problem-oriented patterns to analyze security requirements from an attacker's point of view (Lin et al., 2003). An anti-requirement is fulfilled when a threat initiated by an attacker is realized. Domains are considered as assets. Abuse frames consider problems in isolation, whereas we consider the system as a whole.

Haley et al. provide a problem-oriented framework to analyze security requirements (Haley et al., 2008). Using so called satisfactory arguments, the authors analyze a system with regard to security, based on its functional requirements. There is no specific way to identify threats. Instead, the framework allows to analyze the system as a whole, i.e. to analyze functional requirements, assets, threats, and countermeasures in combination.

In the context of goal-oriented software-engineering, Secure Tropos allows modeling and analyzing security requirements along with functional requirements (Mouratidis and Giorgini, 2007). Another goal-oriented approach is the usage of anti-models (van Lamsweerde, 2004). Currently, we do not consider goals in detail and do not provide any goal refinement process. Instead, we put a special focus on requirements and the corresponding data flow which we analyze with regard to information security. By combining problem frames with goal-oriented methods, it is possible to document the goals of the system and its context (Mohammadi et al., 2013).

Sindre and Opdahl introduce misuse cases as an extension of use cases. A misuse case describes how a software shall not be used, i.e. how potential attackers may harm the software (Sindre and Opdahl, 2005). Misuse cases are a high-level description which does not provide any details about information flows, as we do in our approach.

Microsoft developed a method called STRIDE (Shostack, 2014). 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 threat categories. Each of these categories is a negative counterpart to a security goal. Those categories may be considered as guide words to improve the derivation of threat scenarios for specific statements in our approach.

The STORE method allows to elicit security requirements in the earliest phases of a software development process (Ansari et al., 2019). The authors follow a similar approach to first identify domain knowledge such as external entities and vulnerabilities. To

identify threats, the authors propose a threat dictionary which contains a collection of previously identified threats. Currently, we do not consider any catalogs of threats as an input for our method, but the authors do not consider dependencies between requirements or any information flow. Thus, both approaches may benefit from each other.

Beckers proposes attacker templates to elicit possible attackers to a system (Beckers, 2015). Those templates help to systematically identify and specify the behavior and capabilities of potential attackers. By embedding those templates in our process, we can improve the identification of attackers, and therefore the identification of security incidents.

## 7 CONCLUSION

Finally, we summarize our contributions and provide an outlook on future research directions.

**Summary.** In this paper, we proposed a model-based method to systematically derive relevant security incidents from functional requirements. As required input, we consider a problem-oriented requirements model which we extend with additional domain knowledge, i.e. additional stakeholders and possible attackers. By traversing the requirements model and identifying sequences of statements, our method derives security incidents that may lead to harm for an asset. Since we do not inspect each requirement or assumption in isolation, we consider the software to be developed and its environment as a whole. To further refine the identified incidents, we inspect each requirement and elicited domain knowledge in more detail. Our model-based approach ensures consistency and traceability through all the steps of the method. By providing validation conditions for each step, we assist security engineers in detecting errors in the application of our method as early as possible.

To limit the effort to apply our method and to ensure scalability, we provide a tool for our method. The provided wizards guide through the method, and with our notation, we present the diagrams in a user-friendly way. We formalized several validation conditions to validate the models automatically.

**Outlook.** As future work, we plan to evaluate the effectiveness of our method and our tool. To do so, we cooperate with an industrial company. In an experiment, we will compare the results achieved with our method with the results manually achieved by their security experts.

As mentioned in Section 1, we proposed a template to specify threats. We will embed this template into our method to further assist security engineers in

identifying relevant threats.

To enhance the user experience with our tool, we plan to add quick fixes that help users in correcting errors in their models. Those quick fixes will be based on the not yet completed set of validation conditions.

## REFERENCES

- Ansari, M. T. J., Pandey, D., and Alenezi, M. (2019). STORE: security threat oriented requirements engineering methodology. *CoRR*, abs/1901.01500.
- Beckers, K. (2015). *Pattern and Security Requirements - Engineering-Based Establishment of Security Standards*. Springer.
- FIRST.org (2015). Common Vulnerability Scoring System v3.0: Specification Document. <https://www.first.org/cvss/cvss-v30-specification-v1.8.pdf>.
- Haley, C. B., Laney, R. C., Moffett, J. D., and Nuseibeh, B. (2008). Security requirements engineering: A framework for representation and analysis. *IEEE Trans. Software Eng.*, 34(1):133–153.
- International Organization for Standardization (2018). ISO 31000:2018 Risk management – Principles and guidelines. Standard.
- Jackson, M. A. (2000). *Problem Frames - Analysing and Structuring Software Development Problems*. Pearson Education.
- Lin, L., Nuseibeh, B., Ince, D. C., Jackson, M., and Moffett, J. D. (2003). Analysing security threats and vulnerabilities using abuse frames.
- Lund, M. S., Solhaug, B., and Stølen, K. (2011). *Model-Driven Risk Analysis - The CORAS Approach*. Springer.
- Meis, R. (2013). Problem-based consideration of privacy-relevant domain knowledge. In Hansen, M., Hoepman, J., Leenes, R. E., and Whitehouse, D., editors, *Privacy and Identity Management for Emerging Services and Technologies - 8th IFIP International Summer School, Nijmegen, The Netherlands, June 17-21, 2013, Revised Selected Papers*, volume 421 of *IFIP Advances in Information and Communication Technology*, pages 150–164. Springer.
- Mohammadi, N. G., Alebrahim, A., Weyer, T., Heisel, M., and Pohl, K. (2013). A framework for combining problem frames and goal models to support context analysis during requirements engineering. In Cuzocrea, A., Kittl, C., Simos, D. E., Weippl, E. R., and Xu, L., editors, *5th International Cross-Domain Conference, CD-ARES 2013, Regensburg, Germany, September 2-6, 2013. Proceedings*, volume 8127 of *LNCS*, pages 272–288. Springer.
- Mouratidis, H. and Giorgini, P. (2007). Secure tropos: a security-oriented extension of the tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, 17(2):285–309.
- Shostack, A. (2014). *Threat Modeling: Designing for Security*. Wiley.
- Sindre, G. and Opdahl, A. L. (2005). Eliciting security requirements with misuse cases. *Requir. Eng.*, 10(1).
- Stevens, R., Votipka, D., Redmiles, E. M., Ahern, C., Sweeney, P., and Mazurek, M. L. (2018). The battle for new york: A case study of applied digital threat modeling at the enterprise level. In Enck, W. and Felt, A. P., editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, pages 621–637. USENIX Association.
- van Lamsweerde, A. (2004). Elaborating security requirements by construction of intentional anti-models. In Finkelstein, A., Estublier, J., and Rosenblum, D. S., editors, *26th International Conference on Software Engineering (ICSE 2004), 23-28 May 2004, Edinburgh, United Kingdom*, pages 148–157. IEEE Computer Society.
- van Lamsweerde, A. (2009). *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley.
- Wirtz, R. and Heisel, M. (2018). A systematic method to describe and identify security threats based on functional requirements. In Zemmari, A., Mosbah, M., Cuppens-Boulahia, N., and Cuppens, F., editors, *Risks and Security of Internet and Systems - 13th International Conference, CRISIS 2018, Arcachon, France, October 16-18, 2018, Revised Selected Papers*, volume 11391 of *LNCS*, pages 205–221. Springer.
- Wirtz, R. and Heisel, M. (2019). RE4DIST: model-based elicitation of functional requirements for distributed systems. In van Sinderen, M. and Maciaszek, L. A., editors, *Proceedings of the 14th International Conference on Software Technologies, ICSOFT 2019, Prague, Czech Republic, July 26-28, 2019.*, pages 71–81. SciTePress.
- Wirtz, R., Heisel, M., Borchert, A., Meis, R., Omerovic, A., and Stølen, K. (2018). Risk-based elicitation of security requirements according to the ISO 27005 standard. In Damiani, E., Spanoudakis, G., and Maciaszek, L. A., editors, *Evaluation of Novel Approaches to Software Engineering - 13th International Conference, ENASE 2018, Funchal, Madeira, Portugal, March 23-24, 2018, Revised Selected Papers*, volume 1023 of *Communications in Computer and Information Science*, pages 71–97. Springer.